OPTIMIZATION OF AUTOMATED FLOAT GLASS LINES

A Thesis
Presented to
The Academic Faculty

by

Byungsoo Na

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Industrial and Systems Engineering

Georgia Institute of Technology
May 2011

OPTIMIZATION OF AUTOMATED FLOAT GLASS LINES

Approved by:

Professor Shabbir Ahmed, co-advisor
School of Industrial and Systems Engineering
Georgia Institute of Technology

Professor George Nemhauser, co-advisor
School of Industrial and Systems Engineering
Georgia Institute of Technology

Professor Joel Sokol, co-advisor
School of Industrial and Systems Engineering
Georgia Institute of Technology

Professor Gary Parker
School of Industrial and Systems Engineering
Georgia Institute of Technology

Professor Mark Ferguson
College of Management
Georgia Institute of Technology

Date Approved: 15 December 2010

In loving memory of my grandmother...

ACKNOWLEDGEMENTS

It was a great honor to work with Dr. Shabbir Ahmed, Dr. George Nemhauser, and Dr. Joel Sokol. I am grateful to have these incredible researchers and concerned people as my advisors. I believe that they are the three best advisors at ISyE. As a researcher, I was deeply inspired by their passion toward research and knowledgeable insight. As a person, their great personalities positively influenced on my life. Moreover, I appreciate that they gave me a chance of studying a real-world problem which I am interested in, and it was enjoyable research. Also, I would like to thank them for reviewing the papers line by line with patience and giving me helpful advice. I will miss all the weekly meetings that we had. The meetings were always challenging, but invaluable experiences to me though I had to spend many sleepless nights to prepare for them.

I would like to express my appreciation to the remaining members of my committee, Dr. Gary Parker and Dr. Mark Ferguson. In addition, I am grateful to the AGC company and Mr. Kent Hartsock. They provided me with an interesting topic and financially supported my research.

During my studies, I was extremely fortunate to have interacted with a number of friends and colleagues at ISyE. Special thanks to Dimitri and Akshay for the countless times they helped me throughout the years and for being good friends. I am also indebted to many of Korean friends at ISyE: SooHyun, KySang, TaeSoo, MinKyu, Dong-gu, WonKi, NamSoo and Judy. Their friendship and encouragement made Atlanta feel like home and life at Georgia Tech much more enjoyable.

I would like to thank my parents and my sisters. Without their unconditional love and support, I would not be here. There are no words to express my gratitude to them. Finally, to my wonderful wife, HyoIm and my precious son, JunSeok. At every moment, I thank both of you for being with me. I love you so much.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

In automated float glass manufacturing, a continuous ribbon of glass is cut according to customer orders and then offloaded. We consider the problem of laying out and sequencing the orders on the ribbon so as to minimize waste. The float glass manufacturing process has two types of wasted glass. *Layout scrap* is caused due to suboptimal layout of the rectangular plates on the ribbon and it is similar to scrap that appears in two dimensional cutting problems. The relation between cutting and offloading operations creates a new type of scrap. This type of scrap occurs if some glass is cut but cannot be picked up before it gets to the end of the conveyor because the offloaders are all busy. This wasted glass is called *cycle time scrap*. Thus, given a set of jobs, each specifying a size of glass and a number of units of that size, the *float glass problem* optimizes the sequence and layout of the glass being cut so that the total amount lost to cycle time scrap and layout scrap is minimized.

Traditionally, offloaders were humans, and if too much cycle time scrap was incurred, it was simple to just hire more people, so the float glass problem was less important. However, with manufacturers switching to robotic offloaders, the cost of adding an additional offloader could be in the millions of dollars, so finding good solutions to the float glass problem becomes very important.

The float glass problem has several operational restrictions that are related to other problems such as the guillotine version of the two-dimensional cutting-stock problem, two-stage hybrid flow shops, and cyclic scheduling. However, each of these problems concerns individual aspects of float glass manufacturing, and as far as we know there is no scientific literature on optimal schedules of float glass manufacturing. We propose a two-phase approach to the real-world float glass scheduling problem. In the first phase, called snap construction, we create a *standard snap* for each order by selecting the number and orientation of plates for the order. In the second phase, we determine cutting and offloading

schedules, which we call the *float glass scheduling problem (FGSP)*. We describe the main component of a schedule in FGSP, called *coveys*, and explain how this structure can be used to represent the solution space in a way that aids us to in creating an effective sequencing and assignment algorithm.

Since FGSP is complicated, to obtain analytical results we need to consider simpler versions of FGSP. FGSP has three main elements that make it interesting. Jobs require different processing times, different widths and different numbers of units to be produced. By relaxing the elements, we obtain three simple models: *time*, *unit*, and *width*. For each model, we provide either a polynomial time algorithm or a proof of NP-completeness. With regard to the time model, we provide an $O(n \log n)$ time algorithm, when the number of machines (m) is two and the number of jobs is n. When $m = 3$, we prove that the time model is NP-complete. The unit model is NP-complete by reduction from the parallel machine scheduling problem. We show that the width model can be solved polynomially in $O(nm)$ time by a dynamic programming algorithm. The full model including all three elements is NP-complete since the unit model is NP-complete. Hence, FGSP is hard to solve to optimality in practice, and thus it is important to consider heuristic solution methods. We propose the *Longest Unit First (LUF)* algorithm, and we give a worst case performance bound of $\left\{\left(1 + \frac{m-1}{m}\right) + \left(\frac{1}{3} - \frac{1}{3m}\right)\right\}$; when the number of machines is two, it is $\frac{5}{3}$.

We propose two different methods for snap construction. The method of *selection with a smaller variance* helps to reduce layout scrap while the method of *selection considering machine balancing using mixed integer programming (MIP)* helps to reduce cycle time scrap. After snap construction, we apply two main approaches to solve cutting and offloading schedules: an MIP approach and a heuristic approach. First, our MIP approach is based on a snap discretization formulation, similar to that commonly used in scheduling theory. However, this type of formulation has a huge number of variables and constraints and thus requires a huge running time to solve it. To reduce running time, we give a rolling horizon procedure and a reduced snaps procedure. To improve the yield ratio (ratio of glass input and output), we propose early start of long jobs, long horizon for the last stage, and many jobs in the last stage methods. Second, our heuristic approaches consist of two

construction algorithms, and two improvement algorithms (by local search and by dynamic programming). Our local search algorithms such as *relocation*, *exchange*, and *push-back* effectively work for improving the quality of solutions and the overall heuristic approaches achieve schedules with over 99% yield ratio. In addition to solving cutting and offloading schedules, we conduct sensitivity analysis on the number of offloading machines. Finally, we analyze the ratios of layout scrap and cycle time scrap. In the current manufacturing environment with a sufficient number of offloading machines, most of the scrap is layout scrap. However, when there are fewer offloading machines, the ratio of cycle time scrap increases.

This thesis has two main contributions. (i) Solving the real-world problem: We introduce a new modeling structure, coveys, and apply several optimization methods such as MIP, local search and dynamic programming. Our solution approach produces manufacturing yields greater than 99%; current practice is about 95%. This is a significant improvement and these high-yield solutions can save millions of dollars. (ii) Theoretical analysis: We introduce FGSP, which combines aspects of traditional cutting problems and traditional scheduling problems. We analyze its complexity and give a LUF heuristic whose worst case performance is $\frac{5}{3}$ when $m = 2$.

CHAPTER I

INTRODUCTION

Flat glass is a $53 billion per year industry worldwide [8], with almost all flat glass products being manufactured on float glass lines. New technologies allow float glass manufacturers to increase the level of automation in their plants. However, the question of how to effectively use the automation has given rise to a new and difficult class of optimization problems that have not yet been studied in the literature. These optimization problems combine aspects of traditional cutting problems and traditional scheduling and sequencing problems. In so far as we know, the combination of cutting and scheduling has not been modeled, or solved. In this research, we define and model the problem, and we analyze the complexity of the problem. In addition, we provide several solution methods to maximize manufacturing yields.

In this chapter, we first explain float glass manufacturing process, which motivates this research. We address cutting and scheduling issues occurring in float lines and introduce the problem of optimizing laying out plates and sequencing of jobs so as to maximize the yield ratio. Then, we review related literature and outline the rest of the thesis.

1.1 Float Glass Manufacturing

Float glass manufacturing, as shown in Figure 1.1, is a continuous process whereby a ribbon of molten glass is produced in a furnace and then cooled on a bath of molten tin to ensure flatness. The continuous glass ribbon is then carried on rollers through an annealing lehr, machine-cut according to customer size requirements, and offloaded for storage and distribution. The equipment in this process, beginning with the furnace and ending with the offloading equipment, constitutes a float line.

Assuming that any glass cut could easily be offloaded, the problem of fitting all of the customers' orders into the smallest-possible piece of ribbon is very similar to traditional

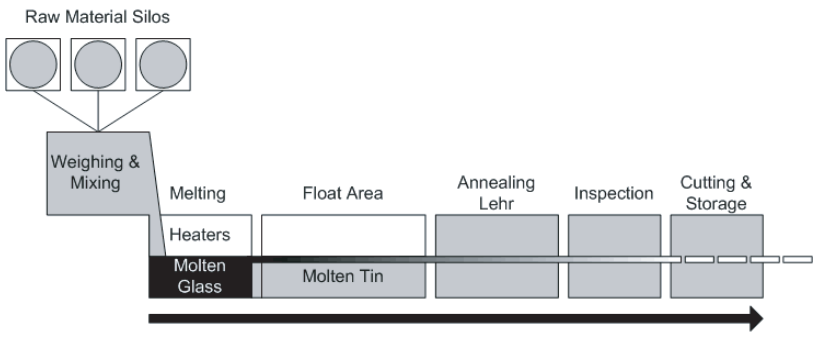


Figure 1.1: Basic float glass manufacturing process [17]

2-dimensional cutting problems. In this case, the only way glass is wasted (other than breakage) is in the process of laying out orders on the ribbon. This wasted glass, or scrap, is known as *layout scrap*.

Our work is motivated by a problem faced by a specific float glass manufacturer that has recently fully automated their previously-manual offloading process. In the float line, there are clear safety and cost advantages to performing offloading with robots rather than humans, but the automation is more restricted in the amount of glass that can be offloaded per unit time. As a result, some additional glass might be wasted if it is cut but cannot be picked up before it gets to the end of the conveyor. This additional scrap is known as *cycle time scrap* because it is caused by the cycle time (minimum time between pickups) of the robots. Of course, cycle time scrap could be eliminated by purchasing more automated equipment, but each robot costs millions of dollars. It is preferable to optimize the sequence and layout of the glass being cut so that the total amount lost to cycle time scrap and layout scrap is small.

1.1.1 Basic Terminology

Our problem concerns processing a given set of customer orders (usually about 60) over a 24-hour working shift. Each customer order consists of a specified number of identical pieces of glass (*plates*) of specified dimension (length $\times$ width $\times$ thickness). For example, a customer might order 200 plates with dimensions 20 inches by 40 inches by 1 inch. Because it is difficult and costly to change the thickness of the ribbon, each production shift is

typically composed of orders of the same thickness.

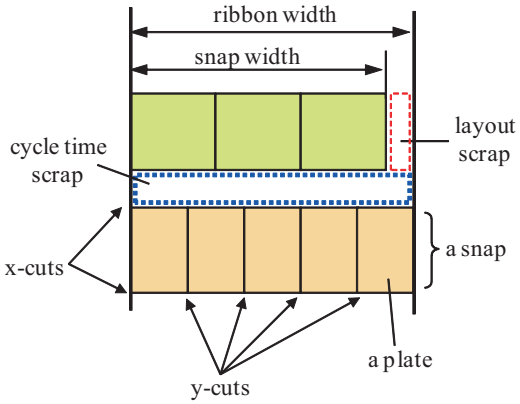
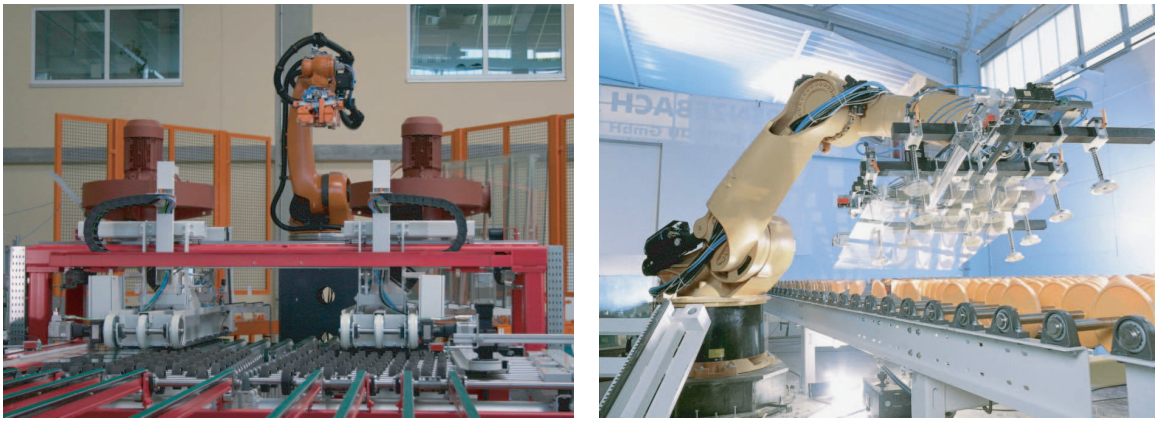


Figure 1.2: Terminology of a float line

Figure 1.2 illustrates terminology of a float glass line. Plates are created by a two-step cutting process in which the ribbon of glass is first scored (etched where plates will be divided), and then snapped along the scores. The *x-cuts* stretch across the width of the ribbon perpendicular to its direction of flow, and the *y-cuts* are at right angles to the x-cuts and stretch between two consecutive x-cuts. The glass between two consecutive x-cuts is referred to as a *snap*, and the *snap time* is the time between the two x-cuts. (Because glass of a given thickness and width is produced at a constant rate, snap time is proportional to the length of glass between the two x-cuts.) Between two consecutive x-cuts, the y-cuts divide a snap into two or more plates. Layout scrap and cycle time scrap are also shown in the figure. Recall that layout scrap is caused by not using the ribbon width fully when laying out the plates, and cycle time scrap is affected by the sequencing of snaps on the ribbon.

After being cut, the plates are offloaded on storage containers by offloading machines. There are several types of offloading machines; this research deals with two common types: *high-speed-stacker* (HSS) and *pick-on-the-fly* (POF) machines. HSS machines simultaneously pick up all the plates from one snap of the ribbon. The requirement for using an HSS machine is that all plates in the snap have the same size and that each plate's size must not be too large. On the other hand, each POF machine can pick up large plates, but it can pick up only one plate at a time. As a result, a snap with m plates requires m POF

machines to be simultaneously available for picking. Figure 1.3 shows photos of HSS and POF machines.



(a) High-speed-stacker (HSS) machine

(b) Pick-on-the-fly (POF) machine

Figure 1.3: Types of offloading machines [11]

1.1.2 Constraints, Objective, and Decisions

Because of the limitations of the glass-making process and the equipment involved, the manufacturer imposes a number of operational restrictions explained below. The first set of restrictions deals with the width and usage of the ribbon.

Continuous Production: Production of glass on the float line is continuous. Therefore, even if no offloading machine is available to pick glass, the glass will still be produced (and therefore wasted as scrap).

Thickness Grouping: Because the thickness of the ribbon is the most difficult attribute to change, many orders of the same thickness are run in one shift. These shifts often take multiple days to complete, and the focus of this paper is on the scheduling within these shifts.

Monotonic Ribbon Width: Every change in ribbon width incurs scrap while the line is being re-adjusted. Therefore, the manufacturer has specified that the ribbon width may only be varied in a monotone way (either non-increasing or non-decreasing) during a shift of a fixed thickness.

Single-Order Snaps: In general cutting-stock problems, a snap may contain different-sized objects in order to make best use of the ribbon. However, in our float glass problem,

all of the plates in a snap must come from the same order. Thus, a solution may not have snaps that contain plates from multiple orders.

Another restriction is related to a product characteristic produced by the float glass manufacturing.

Multi-unit Products: We consider a manufacturing plant with large-scale production capability. Customer orders usually require a large volume of glass (about 100 - 2,000 units). Each order has multiple units.

Some restrictions related to the offloading machines also exist.

Machine Dedication: A float line has a limited number of offloading machines, and each offloading machine can deal with only one container of glass at a time. Since each container stores only one order's glass, all units of an order must be assigned to the same offloading machine.

No Preemption: Once an offloading machine begins to process an order, it must complete all of the units of the order before it can begin to process another order.

Constant Machine Cycle Time: Each offloading machine has a required cycle time: the time it takes to pick a plate, put it into the container, and return to the ready position. The cycle time of offloading machines can vary slightly by machine type and according to the size of plates being picked up. However, the difference is negligible in practice.

Within a shift, a set of customer orders is given, with each order specifying the size of plates and the number of plates to be produced. The measure of a schedule is the amount of glass required to produce all of the given set of orders. We quantify this measure by defining the *yield ratio* as:

$$\text{yield ratio} \quad = \quad 1 - \frac{(\text{cycle time scrap}) + (\text{layout scrap})}{(\text{total glass required for the shift})}.$$

The objective of the float glass problem is to maximize the yield ratio subject to the restrictions already mentioned. To produce a schedule that maximizes yield, we need to determine:

(1) the construction of snaps by laying out the plates in the given orders,

(2) the sequence in which the snaps are processed (i.e., cut and offloaded),

(3) the assignment of plates and/or snaps to offloading machines.

1.2 Literature Review

Although the literature addresses the individual aspects of the float glass problem, such as layout scrap and cycle time scrap, we are unaware of any model in the literature that deals with the full complexity of the problem.

The float glass problem resembles a guillotine version of the two-dimensional cutting-stock problem (2D CSP) studied by Gilmore and Gomory [6, 7] and Beasley [1]. However, the 2D CSP addresses only layout scrap. The float glass problem has the added complexity of cycle time scrap caused by the relation between the cutting and offloading operations. Also, we observe that the 2D CSP is stationary cutting; no time restriction exists to cut rectangular plates. However, the float glass problem has a time restriction; since ribbon is continuously produced and moving on the roller, we have limited time to cut plates before they reach the end of conveyor line. Thus, the float glass problem concerns scheduling as well as cutting.

A related scheduling model is a two-stage hybrid flow shop (HFS) scheduling with no intermediate storage and identical parallel machines in the second stage. The cutting and offloading operations in the float glass problem make up the two stages of a flow shop. See Linn and Zhang [13] for general HFS scheduling, and see Gupta and Tunc [10] for the HFS with parallel machines at the second stage. Also, see Pinedo [15] for the flow shop with limited intermediate storage. Its scheduling notation is $F_2 \mid \text{no wait}, m_1 = 1, m_2 = m \geq 2 \mid C_{\max}$ where m is the number of parallel machines at stage 2. However, HFS scheduling does not address layout scrap. Also, the float glass problem has additional restrictions: constant processing time in stage 2, *machine dedication* and *no preemption* with respect to the multi-units of a given job (otherwise each unit could be considered as an individual job).

Worst case performance analysis has been studied for no-wait (or blocking) flowshops with parallel machines. Sriskandarajah [16] proved that a list scheduling algorithm has a worst case bound of $3 - \frac{1}{m}$ for $F_2|\text{no-wait}, m_1 = 1, m_2 = m \geq 2|C_{\max}$. In the list scheduling algorithm, jobs are scheduled in the order in which they appear on the given list. If jobs are scheduled in non-increasing order of the processing times at the second stage, Sriskandarajah proved that the worst case bound is two.

Other related literature concerns *cyclic scheduling*. When a set of jobs is produced in a no-wait flowshop and each job has multiple units, the same schedule is repeated over and over again. This repeated pattern is called a *cyclic schedule* in operations research (see Pinedo [15] for general cyclic schedules). McCormick et al. [14] studied cyclic schedules in no-wait flow lines or flowshops with blocking where each job type has multiple units to be produced. One cycle of production consists of a set of items and this cycle repeats several times. The sum of processing times in a cycle is called *cycle time*. Since minimizing cycle time is equivalent to maximizing throughput, they proposed heuristics for finding a cyclic schedule so as to minimize cycle time. In chemical engineering processes, Birewar and Grossmann [2] studied cyclic schedules in multiproduct batch plants. They called a cyclic schedule a *campaign*, and compared single-product campaigns with mixed-product campaigns. A MIP model was proposed to minimize cycle time. Birewar and Grossmann [3] also proposed a model designed specifically to handle zero-wait cases.

The float glass problem also yields cyclic schedules, but because of additional restrictions the type of cycles that appear in our research have a different structure than those considered previously. In general cyclic schedules, each machine in a flow line produces several jobs alternately. By the machine dedication and no preemption properties, this is not allowed in our problem, where each machine in the second stage produces only unit of a job until it finishes all of the units of the job. In addition, in cyclic schedules, each cycle may have more than one unit of a job according to the ratio of the number of jobs to be produced. However, in our problem, each cycle has only one unit of a job since a job must be assigned to just one machine at the second stage.

Dash et al. [4] studied the problem of producing rectangular plates for a steel company

to minimize scrap, but this work does not consider the sequencing aspects of our problem.

Thus, none of these earlier works adequately captures the full complexity of the float glass problem.

1.3 Thesis Outline

The ultimate goal of this research is to solve the real-world float glass problem. We provide a two-phase approach for this problem: *snap construction* and *constructing cutting and offloading schedules*. In the following section, we briefly explain *snap construction*, which determines the construction of snaps by laying out the plates in the given orders.

The main challenging issues of this research appear in the second phase problem, which determines the sequence in which the snaps are processed (i.e., cut and offloaded), and the assignment of plates and/or snaps to offloading machines. In order to approach the phase 2 problem analytically, in Chapter II we define a *float glass scheduling problem (FGSP)*. FGSP contains all the restrictions of cutting and offloading processes in a real-world float glass problem except that FGSP assumes identical machines. Then, we describe the main structure of the schedule in FGSP, called *coveys*, and explain how this structure can be used to represent the solution space in a way that motivates us the sequencing and assignment algorithms.

In Chapter III, we analyze FGSP and consider its complexity. FGSP has three main elements that make it interesting. Jobs require different processing times, different widths and different numbers of units (snaps) to be produced. By relaxing the elements, we obtain simple models: the *time* model is related to cycle time scrap, the *width* model is related to layout scrap, and the *unit* model is related to the *end effect*, which will be explained in Chapter V. For each model, we provide either a polynomial time algorithm or a proof of NP-completeness. Since the unit model is NP-complete, the full model including all three elements is NP-complete. Hence, FGSP is hard to solve to optimality in practice, and thus it is essential to consider a heuristic solution method. In Chapter IV, we propose a heuristic algorithm, *Longest Unit First (LUF)* algorithm, and analyze its worst case performance in

terms of quality of solutions.

In Chapter V, we propose solution approaches to the real-world float glass problem. Two methods are proposed for the snap construction phase, and each of them focuses on reducing layout scrap and cycle time scrap, respectively. In constructing cutting and offloading schedules, we propose MIP and heuristic approaches and show computational results. In addition, to find an optimal number of offloading machines, we conduct sensitivity analysis on the number of offloading machines. Chapter VI gives conclusions.

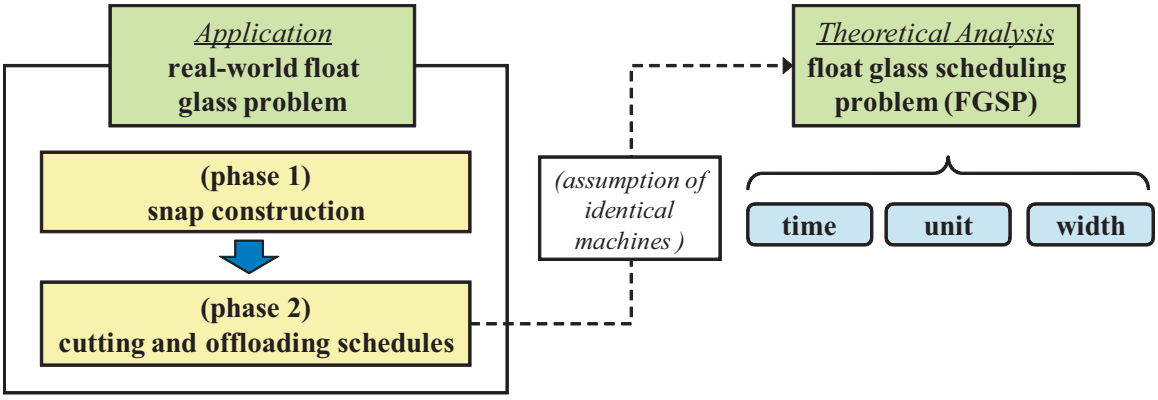


Figure 1.4: Overview of the approach to the float glass problem

Figure 1.4 shows an overview of our approach. For the real-world problem, we present a two-phase approach that is discussed in Chapters I and V. To approach the second phase problem of constructing cutting and offloading schedules, we introduce and analyze FGSP. Chapters II, III and IV deal with the theoretical analysis of FGSP.

1.4 Two-phase Approach

We can define the overall systems problem of maximizing the yield ratio in which we need three sets of decisions: (1) the construction of snaps by laying out the plates, (2) the sequence of cutting and offloading, and (3) the assignment of plates and/or snaps to offloading machines. However, we separate determining the layout of plates into a preprocessing step. Since the float line has flexibility in changing ribbon width, we can easily construct each snap with a small amount of layout scrap. Therefore, in float glass manufacturing, the

amount of scrap incurred from laying out rectangular plates is quite small and this aspect of the problem is not difficult. Hence, we briefly explain the problem of laying out plates and then mainly discuss the scheduling problem of sequencing and assignment.

1.4.1 Snap Construction

The problem of laying out plates is referred to as *snap construction*. We create the *standard snap* for each order by selecting the number and orientation of plates in the order's snaps. There are only a few choices for each order. For example, to produce 1,200 plates of size of 35×40 on a ribbon with minimum width of 100 and maximum width of 140, we can produce three plates or four plates of dimensions 35×40, or three plates of dimensions 40×35 (see Figure 1.5). In our solution approaches, we will provide certain rules of constructing standard snaps in order to maximize the yield ratio rather than only to minimize scrap incurred from laying out plates. Once such a standard snap is created, each order is filled by repeatedly cutting its standard snap. Suppose that the standard snap in the example is four plates of dimensions 35×40. Now, we produce 300 units of this standard snap with the width of 140. Such a task consisting of multiple units of a standard snap is referred to as a *job* in this research.

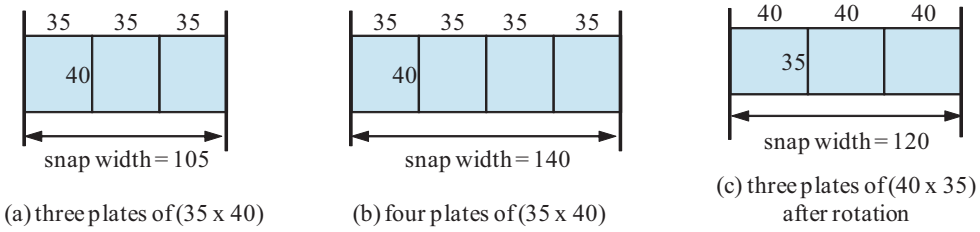


Figure 1.5: Snap construction alternatives

1.4.2 Constructing Cutting and Offloading Schedules

Once standard snaps are created, we construct cutting and offloading schedules. This phase of the problem is very complex and can have a huge effect on the yield ratio of the real-world float glass manufacturing. To do a systematic analysis of this phase, we define the *float glass scheduling problem (FGSP)* and provide a solution structure for it in the next chapter.

CHAPTER II

FLOAT GLASS SCHEDULING PROBLEM (FGSP)

In the previous chapter, we proposed a two-phase approach for the real-world float glass problem and we mentioned that the second phase, constructing cutting and offloading schedules, is the main part of the real-world problem. The second phase problem includes both sequencing and assignment issues, which cause layout scrap and cycle time scrap. In this chapter, we define the *float glass scheduling problem (FGSP)* based on the problem of constructing cutting and offloading schedules. FGSP assumes that there exists only one type of machine. We first show that minimizing scrap is equivalent to minimizing total processing time of the cutting machine. We then show that there is an optimal solution with a specific structure which motivates our proposed solution methods. Finally, we provide a formal mathematical description of the FGSP.

2.1 Problem Statement

In float glass manufacturing, there is a set of jobs J to be processed in two stages, stage 1 (the cutting process) and then stage 2 (the offloading process). Stage 1 has one machine and stage 2 has m identical parallel machines. Job j has n_j units that must be produced; in float glass terminology, the number of units corresponds to the number of snaps. The processing time per unit of job j in stage 1 is t_j (the cutting time of job j) and the processing time of a unit of every job in stage 2 is T (by the constant machine cycle time property). No intermediate storage exists between stages 1 and 2. The property of continuous production characterizes no-wait scheduling. In addition, job j has snap width w_j. The objective is to minimize scrap. We call this problem **FGSP** (the *float glass scheduling problem*).

Next we show that FGSP is equivalent to the problem of minimizing the completion time of the cutting machine.

Proposition 1. *Minimizing scrap is equivalent to minimizing the completion time of the cutting machine.*

Proof. Since glass is continuously passing through the cutting machine at a constant rate, the completion time of the cutting machine is the sum of the time for cutting ordered glass and the time for cutting scrap (including both cycle time scrap and layout scrap). The result then follows immediately since the time for cutting ordered glass is a constant and the time for cutting scrap is proportional to the amount of scrap. □

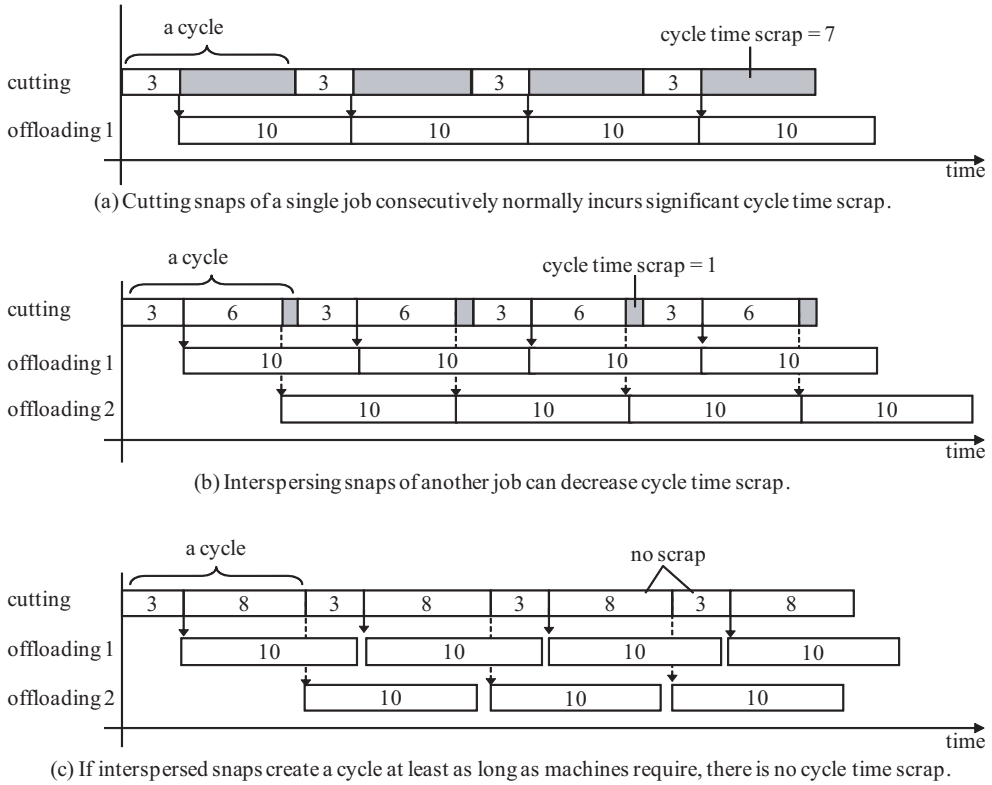
Therefore, in scheduling terminology, FGSP is a no-wait hybrid flowshop problem with parallel machines at stage 2, whose objective is to minimize completion time subject to some additional restrictions: constant processing time in stage 2, *machine dedication* and *no preemption* with respect to the multi-units of a given job. Because of such additional restrictions, we have a new type of schedule structure that is discussed in the following section.

2.2 A Cyclic Schedule Structure: a Covey

Since a set of jobs with multiple units is produced in a no-wait flowshop, FGSP also yields cyclic schedules. However, because of the *machine dedication* and *constant machine cycle time* properties, the type of cycles in FGSP has a different structure than general cyclic schedules. In this section, we describe the main structure of the schedules of FGSP and explain how this structure motivates the development of our proposed sequencing and assignment algorithms.

First, we note that the cutting of snaps requires much less time than the offloading machines' cycle times. (Most snaps are cut in 2.5 to 5 seconds, compared to machine cycle time of about 13 seconds.) Because glass flows through the system continuously, a schedule that cuts one snap of a job after another without any intervening snaps will incur significant cycle time scrap. For example, assume that machine cycle times are 10 seconds and a snap requires 3 seconds of glass. Then, cutting this job's snaps consecutively would require 7 seconds of wasted glass between snaps. That is, the sequence on the ribbon would be: snap

(3 seconds), scrap (7 seconds, until offloading machine is ready), snap (3 seconds), scrap (7 seconds), etc. (see Figure 2.1 (a)). Therefore, good schedules will require snaps of other jobs between two snaps of the same job, to either decrease or eliminate cycle time scrap (see Figures 2.1 (b) and (c)).



(a) Cutting snaps of a single job consecutively normally incurs significant cycle time scrap.

(b) Interspersing snaps of another job can decrease cycle time scrap.

(c) If interspersed snaps create a cycle at least as long as machines require, there is no cycle time scrap.

Figure 2.1: Gantt chart showing how cycle time scrap is incurred

Because offloading machines must finish loading the container(s) of one job before starting the next, a job may not be preempted. Therefore, rotating through the jobs' snaps until one of the jobs finishes (as shown in Figures 2.1 (b) and (c)) is necessary to minimize cycle time scrap. As a result, snaps from different jobs are interspersed with each other. We refer to such a set of jobs being simultaneously processed in rotation as a *covey*.

A schedule can be defined as a sequence of coveys where between two consecutive coveys, one or more jobs have finished and/or one or more jobs have started. The optimization problem thus consists of determining a sequence of coveys that produce all jobs such that the yield ratio is maximized.

Table 2.1 introduces an example set of five jobs. For this example, we assume that two

Table 2.1: Five jobs are to be produced

index	snap time	# of snaps	snap width	plates per snap	offloading machine
A	3.5	150	132	3	HSS 1
B	3.0	200	132	5	HSS 2
C	4.5	100	132	2	POF 1 & 2
D	4.0	100	130	1	POF 1
E	2.0	80	128	7	HSS 1

HSS machines and two POF machines are available and that their cycle time is 10 seconds. In this example, we define three coveys. The first covey (Covey P) consists of Jobs A, B, and C. (This set of jobs uses all four offloading machines, since Job C has two plates per snap and thus requires both POF machines.) After 100 snaps of this covey, Job C is completed and is replaced by Job D. Thus, Jobs A, B, and D define the second covey (Covey Q). After 50 snaps of Covey Q, Job A is completed (a total of 150 snaps) and is replaced by Job E. Jobs E, B, and D thus define the third covey (Covey R). The sequence of snaps produced by the cutter and its relation to the coveys are described in Figure 2.2. The Gantt chart in Figure 2.3 illustrates this schedule for each machine. The top row in the figure shows the schedule for the cutters, and the other four rows show the schedule for each offloading machine.

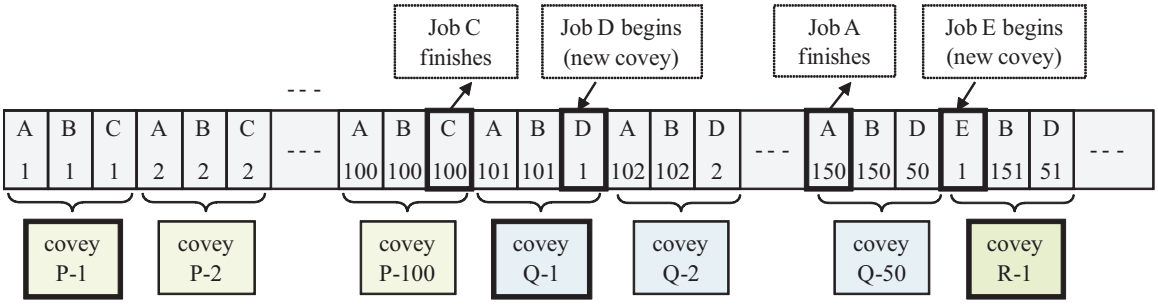


Figure 2.2: Cutting-based view of coveys

Another useful representation of coveys is a snap-based view illustrated in Figure 2.4. This view specifies the sequence of jobs, their assignment to offloading machines, and the number of snaps to be produced of each job and of each covey. In the example, Figure 2.4 shows the assignment of jobs to four machines and the offloading sequence of each machine. - Machine HSS1 first offloads 150 snaps of Job A and then offloads 80 snaps of Job E.

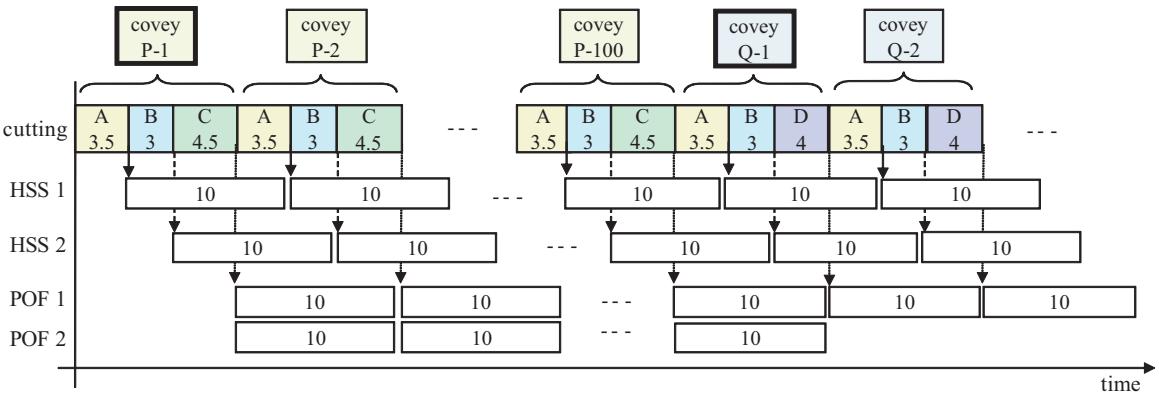


Figure 2.3: Coveys in Gantt chart

- HSS2 offloads 200 snaps of Job B and then remains idle.

- POF1 offloads 100 snaps of Job C and 100 snaps of Job D and then becomes idle.

- POF2 offloads 100 snaps of Job C and afterwards becomes idle for the rest of the time.

In this view, it is easy to see the change of coveys; Covey P runs for 100 snaps, followed by 50 snaps of Covey Q and 50 snaps of Covey R (and then 30 snaps of a fourth covey consisting only of Job E).

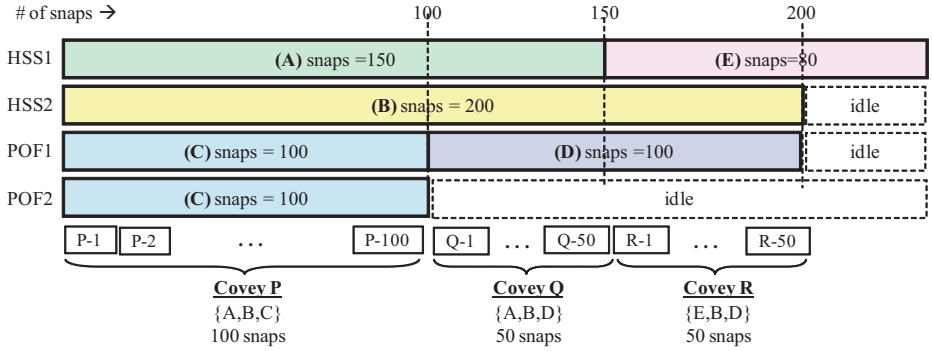


Figure 2.4: Snap-based view of coveys

For each covey, it is easy to calculate the required ribbon width. For ribbons with non-decreasing (non-increasing) width, the ribbon width for a covey is the maximum width of a snap in that covey or any preceding (subsequent) covey.

The volume of glass sent through the system per unit time is constant, as is the length and thickness of a snap. Therefore, for a snap of given length and thickness, the time

required to cut a snap is a function of ribbon width. Consider a snap whose dimensions (length l, width w, and thickness t) is being cut on a line that produces V volume of glass per second. Such a snap requires $d = \frac{lwt}{V}$ seconds of glass on a ribbon of width w. However, if the snap is run on a wider ribbon of width $w' > w$, the snap time increases to $d' = \frac{lw't}{V} = d\frac{w'}{w}$. (Note that although this means wider ribbon widths might help reduce cycle time scrap, they simultaneously increase layout scrap, since $l(w - w')t$ volume of glass is wasted.)

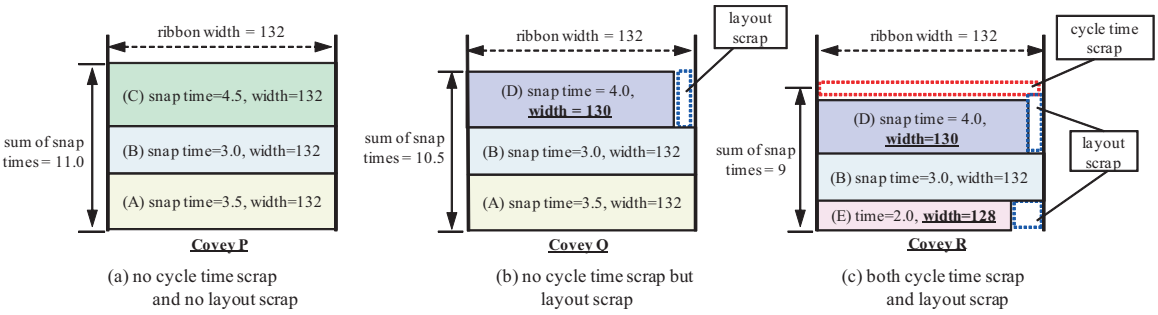


Figure 2.5: Scrap according to different covey construction

Figure 2.5 gives a ribbon-based view of one cycle through each of the coveys P, Q, and R. It shows the amount of scrap incurred in each covey. In Covey P, all three jobs require the same snap width as the ribbon width (132), so this covey has no layout scrap. In addition, the sum of snap times is $3.5 + 3.0 + 4.5 = 11$ seconds, which exceeds the cycle time of the offloading machines. Therefore, Covey P does not have any cycle time scrap. In Covey Q, the snap width of Job D is 130, but the ribbon width must remain 132 because Jobs A and B require a ribbon width of at least. Thus, Covey Q incurs layout scrap corresponding to the area of difference between 132 and 130, multiplied by the ribbon's thickness. The sum of snap times is $3.5 + 3.0 + 4.0 = 10.5 > 10$ seconds, so this covey does not have any cycle time scrap. In Covey R, the snap width of Job E is 128 and that of Job D is 130, but the ribbon width is 132. Therefore, Covey R has layout scrap corresponding to these two areas times the ribbon's thickness. The sum of snap times is $2.0 + 3.0 + 4.0 = 9.0$ seconds, which is less than the offloading machines' cycle time of 10 seconds. Thus, Covey R has cycle time scrap corresponding to the volume of glass produced in one second of idle time.

The examples that we considered are coveys in which each job appears only once. Now,

we consider a covey in which at least one job appears more than once. Such a covey is called a *mixed covey*. The jobs which appear more than once are called *duplicating jobs*. Recall that by the machine dedication restriction, a job that appears more than once must be assigned to the same offloading machine. On the other hand, a covey in which each job appears once is referred to as a *minimal covey*.

A mixed covey can be divided into several minimal coveys in the following way. If a covey has more than one appearance of a job, we split the covey into two at the point where a job is repeated for the first time. We now have a minimal covey and (possibly) a mixed covey. The process is repeated on the remaining mixed coveys. For example, suppose that a mixed covey consists of jobs $x, a_1, a_2, y, a_3, a_4, x, a_5, z, y, a_6$, y, a_7, x, a_8, and z. Then, the first minimal covey consists of jobs x, a_1, a_2, y, a_3, and a_4, and the second minimal covey consists of jobs x, a_5, z, y, and a_6, and the third minimal covey consists of y, a_7, x, a_8, and z. We observe that the first job of each minimal covey appears in the previous minimal covey because of the above dividing rule.

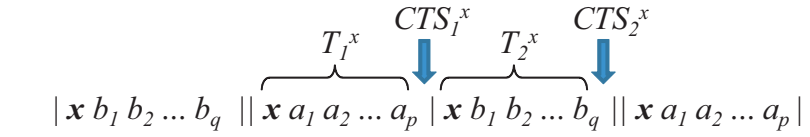
Now, we consider the change of cycle time scrap and layout scrap after converting a mixed covey to minimal coveys.

Proposition 2. *Converting a mixed covey to minimal coveys does not create an additional cycle time scrap if the number of duplicating jobs is no more than two.*

Proof. We first consider a case where the number of duplicating jobs in a mixed covey is one and then a case where the number of duplicating jobs is two.

Case 1: The number of duplicating jobs in a mixed covey is one.

In Figure 2.6, a mixed covey consists of jobs $x, a_1, a_2, \ldots, a_p, x, b_1, b_2, \ldots, b_{q-1}$ and b_q. Only job x is duplicating. Since other jobs are offloaded by different offloading machines, we consider cycle time scrap of job x. Denote the sum of snap times of jobs between the ith appearance of job j and the $(i+1)$th appearance of job j by T_i^j. In addition, denote cycle time scrap caused by the ith and $(i+1)$th appearance of job j by CTS_i^j. In a mixed



(a) A mixed covey consists of jobs $x, a_1, ..., a_p, x, b_1, ..., b_{q-1}$ and b_q.

(b) The first minimal covey consists of jobs $x, a_1, a_2, ..., a_{p-1}$ and a_p.
The second minimal covey consists of jobs $x, b_1, b_2, ..., b_{q-1}$ and b_q.

Figure 2.6: Converting a mixed covey to minimal coveys (one duplicating job)

covey of Figure 2.6 (a), we have

$$T_1^x = t_x + \sum_{h=1}^{p} t_{a_h}, \qquad CTS_1^x = \max\left(T - T_1^x,\ 0\right)$$
$$T_2^x = t_x + \sum_{h=1}^{q} t_{b_h}, \qquad CTS_2^x = \max\left(T - T_2^x,\ 0\right).$$

Therefore, cycle time scrap incurred in one rotation of a mixed covey is

$$CTS_1^x + CTS_2^x \quad = \quad \max\left(T - T_1^x,\ 0\right) + \max\left(T - T_2^x,\ 0\right).$$

The mixed covey can be converted to the first minimal covey with jobs $x, a_1, a_2, \ldots, a_p$ and the second minimal covey with jobs $x, b_1, b_2, \ldots, b_q$ illustrated in Figure 2.6 (b). Denote the sum of snap times of jobs in the ith minimal covey by T_i and cycle time scrap in the ith minimal covey by CTS_i. Then, in the minimal coveys of Figure 2.6 (b), we have

$$T_1 = t_x + \sum_{h=1}^{p} t_{a_h}, \qquad CTS_1 = \max\left(T - T_1,\ 0\right)$$
$$T_2 = t_x + \sum_{h=1}^{q} t_{b_h}, \qquad CTS_2 = \max\left(T - T_2,\ 0\right).$$

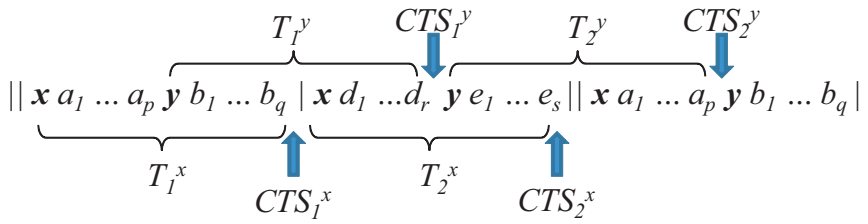
Therefore, the sum of cycle time scrap incurred in one rotation of minimal coveys is

$$CTS_1 + CTS_2 \quad = \quad \max\left(T - T_1,\ 0\right) + \max\left(T - T_2,\ 0\right).$$

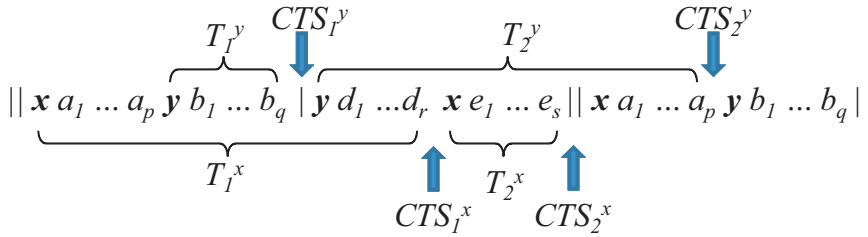
Since $T_1 = T_1^x$ and $T_2 = T_2^x$, cycle time scrap in one rotation of a mixed covey is equal to the sum of cycle time scrap in one rotation of its corresponding minimal coveys. Therefore,

converting a mixed covey to minimal coveys does not create an additional cycle time scrap. When a mixed covey is divided into more than two minimal coveys, we have the same result using similar arguments.

Case 2: The number of duplicating jobs in a mixed covey is two.



(a) Duplicating jobs appear in the order of *x, y, x* and *y*.



(b) Duplicating jobs appear in the order of *x, y, y* and *x*.

Figure 2.7: Cycle time scrap of a mixed covey (two duplicating jobs)

Suppose that jobs x and y are duplicating. There are two subcases illustrated in Figure 2.7.

Subcase 2-1: Duplicating jobs appear in the order of x, y, x and y (Figure 2.7 (a)).

First, in a mixed covey of Figure 2.7 (a), we have

$$CTS_1^x = \max\left(T - T_1^x - CTS_2^y,\ 0\right), \qquad CTS_1^y = \max\left(T - T_1^y - CTS_1^x,\ 0\right),$$
$$CTS_2^x = \max\left(T - T_2^x - CTS_1^y,\ 0\right), \qquad CTS_2^y = \max\left(T - T_2^y - CTS_2^x,\ 0\right).$$

In the corresponding minimal coveys, we have

$$\begin{aligned} CTS_1 &= \max\left(T - T_1,\ 0\right) \\ &= \max\left(T - T_1^x,\ 0\right) \qquad (\because T_1 = T_1^x) \\ &\le \max\left(T - T_1^x - CTS_2^y,\ 0\right) + CTS_2^y \quad (\because \max(A,0) \le \max(A-B,0) + B,\ B \ge 0) \\ &= CTS_1^x + CTS_2^y \end{aligned}$$

$$
\begin{aligned}
CTS_2 &= \max\big(T - T_2,\ 0\big) \\
&= \max\big(T - T_2^x,\ 0\big) \qquad (\because T_2 = T_2^x) \\
&\le \max\big(T - T_2^x - CTS_1^y,\ 0\big) + CTS_1^y \quad (\because \max(A,0) \le \max(A-B,0) + B,\ B \ge 0) \\
&= CTS_2^x + CTS_1^y.
\end{aligned}
$$

Therefore, it follows that

$$
\begin{aligned}
&\text{(the sum of cycle time scrap incurred in one rotation of minimal coveys)} \\
&= CTS_1 + CTS_2 \\
&\le CTS_1^x + CTS_2^y + CTS_2^x + CTS_1^y \\
&= \text{(cycle time scrap incurred in one rotation of a mixed covey).}
\end{aligned}
$$

Therefore, converting a mixed covey to minimal coveys does not create an additional cycle time scrap. When a mixed covey is divided into more than two minimal coveys, we have the same result using similar arguments.

Subcase 2-2: Duplicating jobs appear in the order of x, y, y and x (Figure 2.7 (b)).

First, in a mixed covey of Figure 2.7 (b), we have

$$
\begin{aligned}
CTS_1^x &= \max\big(T - T_1^x - CTS_1^y - CTS_2^y,\ 0\big), \qquad CTS_1^y = \max\big(T - T_1^y,\ 0\big), \\
CTS_2^x &= \max\big(T - T_2^x,\ 0\big), \qquad CTS_2^y = \max\big(T - T_2^y - CTS_1^x - CTS_2^x,\ 0\big).
\end{aligned}
$$

In the corresponding minimal coveys, we have

$$
\begin{aligned}
CTS_1 &= \max\big(T - T_1,\ 0\big) \\
&\le \max\big(T - T_1^y,\ 0\big) \qquad (\because T_1 > T_1^y) \\
&= CTS_1^y \\
CTS_2 &= \max\big(T - T_2,\ 0\big) \\
&\le \max\big(T - T_2^x,\ 0\big) \qquad (\because T_2 > T_2^x) \\
&= CTS_2^x.
\end{aligned}
$$

Therefore, it follows that

$$\begin{aligned} & \text{(the sum of cycle time scrap incurred in one rotation of minimal coveys)} \\ = \;& CTS_1 + CTS_2 \\ \leq \;& CTS_1^x + CTS_1^y + CTS_2^x + CTS_2^y \\ = \;& \text{(cycle time scrap incurred in one rotation of a mixed covey).} \end{aligned}$$

When a mixed covey is divided into more than two minimal coveys, we have the same result using similar arguments. □

Proposition 3. *Converting a mixed covey to minimal coveys does not create an additional layout scrap if a mixed covey is divided into two minimal coveys.*

Proof. Assume that mixed covey R consists of jobs $A_1, A_2, \ldots, A_m, B_1, B_2, \ldots, B_n$ where at least one duplicating job exists. Then, mixed covey R can be divided into (minimal) covey P with jobs $A_1, A_2, \ldots, A_m$ and (minimal) covey Q with jobs $B_1, B_2, \ldots, B_n$. Now, consider the change of layout scrap after converting a mixed covey to minimal coveys. The amount of layout scrap can be different depending on ribbon width. Let w_{A_i} be the width of job A_i, and $w(P), w(Q)$ and $w(R)$ represent the ribbon width of covey P, covey Q, and the mixed covey R, respectively. Then,

$$\begin{aligned} w(P) &= \max\{w_{A_1}, w_{A_2}, \ldots, w_{A_m}\} \\ w(Q) &= \max\{w_{B_1}, w_{B_2}, \ldots, w_{B_n}\} \\ w(R) &= \max\{w_{A_1}, \ldots, w_{A_m}, w_{B_1}, \ldots, w_{B_n}\} \\ &= \max\{w(P), w(Q)\}. \end{aligned}$$

The ribbon width of covey P is the maximum snap width among jobs $A_1, A_2, \ldots, A_m$, and that of covey Q is the maximum snap width among jobs $B_1, B_2, \ldots, B_n$. Similarly, the ribbon width of mixed covey R is the maximum snap width among $A_1, A_2, \ldots, A_m, B_1, B_2, \ldots, B_n$,

which is equivalent to the maximum ribbon width of coveys P and Q. Therefore, we have

$$\begin{aligned}
(\text{layout scrap of mixed covey } R) &= \sum_{i=1}^{m} \big(w(R) - w_{A_i}\big) + \sum_{i=1}^{n} \big(w(R) - w_{B_i}\big) \\
&\geq \sum_{i=1}^{m} \big(w(P) - w_{A_i}\big) + \sum_{i=1}^{n} \big(w(Q) - w_{B_i}\big) \\
&= (\text{layout scrap of covey } P) + (\text{layout scrap of covey } Q).
\end{aligned}$$

Hence, the amount of layout scrap of two minimal coveys is no more than that of mixed coveys containing both of them. □

Besides the amount of scrap, when we use minimal coveys, we can finish the offloading operation of covey P earlier, and containers corresponding to covey P can leave earlier for delivery. However, in the mixed covey schedule, all of the containers must wait until all of the jobs of mixed covey R are finished. Hence, in terms of shipping time, minimal coveys are also better than mixed coveys.

Based on the above Propositions 2 and 3, we conjecture that there exists an optimal schedule for FGSP that is a sequence of minimal coveys. For the rest of the thesis, we only consider schedules consisting of minimal coveys.

Transient Scrap

Usually, one job exits from a covey and another job enters the covey. When two or more jobs coincidentally end at the same time and more than one job enters to form the next covey, additional waste can occur in one rotation where the coveys transition. In Figure 2.8, assume that the cycle time of offloading machines is 10 seconds and there are three offloading machines. The first covey consists of jobs A, B, and C, and its sum of snap times is 4+1+5 = 10 (seconds), which does not produce any cycle time scrap. Suppose that jobs A and B are finished at the same rotation, and they are replaced by jobs D and E, respectively. Then, the second covey consists of jobs D, E, and C, and its sum of snap times is 2+3+5 = 10 (seconds), which again has no cycle time scrap. However, during the transition of coveys, two seconds of additional cycle time scrap is incurred. This phenomenon rarely

happens in practice and the amount of scrap is negligible for real-world applications. Thus, we ignore this small amount of transient scrap in our model.

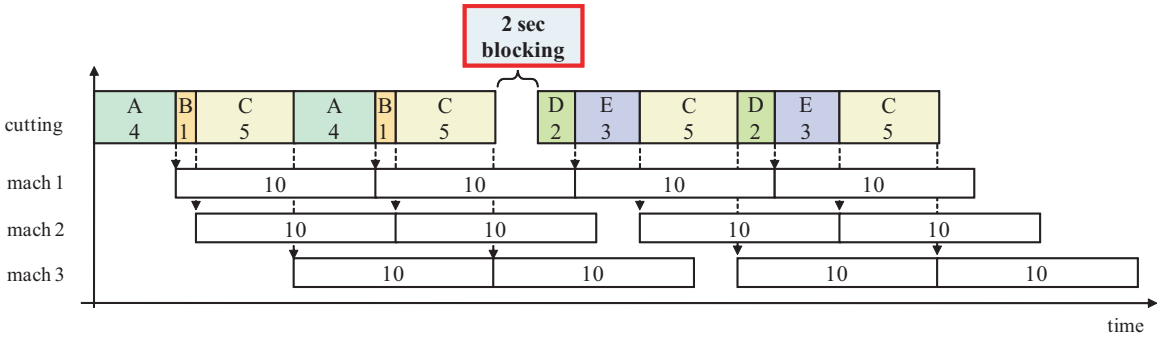


Figure 2.8: Scrap in coveys transition

2.3 Mathematical Description

We denote the kth covey by C_k, where $C_k \subset J$. The number of rotations of the kth covey is $n(C_k)$. The *ribbon width* of covey k, called $w(C_k)$, is defined as the maximum value among widths of the jobs in both covey k and all subsequent coveys:

$$w(C_k) = \max_{j \in C_i:\ \forall i \geq k} w_j.$$

Then, ribbon width is monotonically non-increasing: if $i > j$, then $w(C_i) \leq w(C_j)$. If job j is run on covey k with ribbon width $w(C_k)$, the actual processing time of job j is $\frac{w(C_k)}{w_j} t_j$. We note that the processing time of job j increases by

$$\frac{w(C_k)}{w_j}\ t_j - t_j = \frac{w(C_k) - w_j}{w_j}\ t_j.$$

This amount of time increase corresponds to layout scrap caused by width difference. The processing time of one rotation of covey k, denoted by $t(C_k)$, is defined as the sum of stage 1 processing times of one unit of each job in covey k:

$$t(C_k) = \sum_{j \in C_k} \frac{w(C_k)}{w_j} t_j.$$

Coveys must satisfy the following restrictions:

(C.1) Units of a Job: The number of units of job j equals the number of rotations of the coveys containing job j:

$$n_j = \sum_{k:j \in C_k} n(C_k).$$

(C.2) Limited Number of Machines: The number of jobs in a covey is at most the number of stage 2 machines: $|C_k| \leq m, \ \forall k$. The ith element in a covey is processed by the ith machine ($i \leq m$). If the ith element in a covey is empty, it means that the ith machine is idle when the covey is processed.

(C.3) No Preemption of a Product: If job j is an element of both C_p and C_q ($p < q$), then j must be in C_r for all r such that $p < r < q$.

(C.4) Machine Dedication: If job j is assigned to the ith element of a covey in which job j appears for the first time, then j must be assigned to the ith element of all the following coveys that contain job j.

(C.5) Completion of All Jobs: The set J of jobs must be covered by the elements in all coveys: $\bigcup C_k = J$.

In terms of coveys, the objective function is given by

$$\min C_{\max} \quad = \quad \min \sum_k \ \Big(n(C_k) \times \max\{t(C_k), \ T\} \Big).$$

A *schedule* in FGSP is defined as a sequence of coveys. Once a sequence of coveys is determined, we can easily find implied start and completion times of each job as well as the assignment of jobs to the stage 2 machines.

CHAPTER III

ANALYSIS AND COMPLEXITY OF FGSP

In the previous chapter, we introduced FGSP and provided its underlying structure, *coveys*. FGSP is a complex problem, so to obtain analytical results we need to consider special cases. FGSP has three main elements that make it interesting. Jobs require different processing times in stage 1, jobs require different (snap) widths, and may have a different number of units to be produced. If processing times, widths and number of units are identical for each job, the problem is trivial. We first consider the complexity of simple problems by relaxing each of the three elements of interest. The time model is related to cycle time scrap and the width model is related to layout scrap. The unit model is related to the *end effect*, which will be discussed in Chapter V. Table 3.1 summarizes the complexity of simple models and the full model.

Table 3.1: Summary of complexity

model	complexity	reduction if NP-complete, or algorithm if in P (n is # of jobs and m is # of machines)
time	if $m = 2$, in P if $m = 3$, NP-complete	*match largest and smallest* algorithm: $O(n \log n)$ reduction from 3-PARTITION
unit	if $m \geq 2$, NP-complete	reduction from $P\|\|C_{\max}$
width	if $m \geq 2$, in P	dynamic programming: $O(nm)$
time+unit	if $m \geq 2$, NP-complete	unit model is NP-complete if $m \geq 2$
time+width	if $m = 2$, in P if $m = 3$, NP-complete	minimum weighted matching: $O(n^4)$ time model is NP-complete if $m = 3$
time+unit+width	if $m \geq 2$, NP-complete	unit model is NP-complete if $m \geq 2$

3.1 Time Model

When every job has the same width and requires the same number of units to be produced, but jobs might require different stage 1 processing times per unit, we refer to the model as the *time model*. Using the notation of Garey and Johnson [5], the time model is stated as

follows:

Time Model

INSTANCE: Finite set J, a time $t_j \in \mathbb{R}^+$ for each $j \in J$, a positive integer m (the number of machines), and positive numbers T (the cycle time) and K.

QUESTION: Can J be partitioned into disjoint subsets $C_1, C_2, \cdots$ such that

$$\begin{aligned} |C_i| &\leq m, \quad \forall i \qquad \text{(at most } m \text{ jobs in each covey)} \\ \sum_i \max\Big\{ \sum_{j \in C_i} t_j,\ T \Big\} &\leq K \qquad\qquad \text{(the completion time)} \end{aligned}$$

For $m = 2$, a simple $O(n \log n)$ algorithm solves the time model where n is the number of jobs.

Algorithm: Match Largest and Smallest Jobs

(Step 1) Sort jobs by non-decreasing order in time. Let $(j_1, j_2, \ldots, j_{n-1}, j_n)$ be the sequence of jobs after sorting.

(Step 2-a) If the number n of jobs is even, make coveys $(j_1, j_n), (j_2, j_{n-1}), \ldots, (j_{\frac{n}{2}}, j_{\frac{n}{2}+1})$.

(Step 2-b) If the number n of jobs is odd, make one covey (j_n) for the job with the largest time, and then make coveys for the remaining jobs as in the even case.

Theorem 4. *The algorithm* match largest and smallest jobs *produces an optimal solution in the time model when the number of machines is two.*

Proof. We prove the proposition for an even number $2k$ of jobs; we can reduce the case of an odd number of jobs to the even case by adding a dummy job with $t = 0$. We first prove that an optimal solution exists where every covey contains two jobs, and then we show that the algorithm produces a solution that is at least as good as any other solution with two jobs in each covey, and is thus optimal overall.

To prove that an optimal solution exists where each covey has exactly two jobs, consider an optimal solution S_1 with a covey C_p that consists of one job p with processing time t_p.

Since the number of jobs is even, there exists another covey $C_q \in S_1$ that also consists of one job q with processing time t_q. We can easily construct another solution S_2 that is identical to S_1 except that coveys C_p and C_q are replaced by a single covey C_r containing jobs p and q. The completion time of S_2 is no more than that of S_1:

$$C_{\max}(S_1) - C_{\max}(S_2) \quad = \quad \big\{ \max\{T,\ t_p\} + \max\{T,\ t_q\} \big\} - \max\{T,\ t_p + t_q\} \geq 0.$$

Without loss of generality, assume that the jobs $j_1, \ldots, j_{2k}$ have stage 1 processing times $t_1 \leq t_2 \leq \cdots \leq t_{2k}$. Then, applying the algorithm yields a solution S^* with following coveys: $\{j_1, j_{2k}\}, \{j_2, j_{2k-1}\}, \cdots, \{j_k, j_{k+1}\}$. Each covey consists of two jobs, and the jobs j_i and j_{2k-i+1} are matched.

Consider a solution $S_0 \neq S^*$ with two jobs in each covey, such that the completion time of S_0 is at least as small as that of any other solution with two jobs per covey.

We can transform S_0 into S^* using the following k-step exchange procedure. We begin with S_0. At each step i of the procedure, for $1 \leq i \leq k$, if jobs j_i and j_{2k-i+1} are in the same covey, we leave the solution unchanged. Otherwise, there must be jobs j_{u_i} and j_{v_i} such that the solution contains coveys $\{j_i, j_{u_i}\}$ and $\{j_{v_i}, j_{2k-i+1}\}$. In this case, modify the solution by replacing coveys $\{j_i, j_{u_i}\}$ and $\{j_{v_i}, j_{2k-i+1}\}$ with coveys $\{j_i, j_{2k-i+1}\}$ and $\{j_{u_i}, j_{v_i}\}$. After k steps, we will be left with solution S^*. We note that for each step i where a modification is made, $i < u_i, v_i < 2k - i + 1$ since the procedure guarantees all jobs smaller than i and larger than $2k-i+1$ will already be matched as in solution S^*. Thus, $t_i < t_{u_i}, t_{v_i} < t_{2k-i+1}$.

Below, we prove that no step of the exchange procedure will increase the completion time. Therefore, $C_{\max}(S^*) \leq C_{\max}(S_0)$, so S^* must be an optimal solution.

Claim: No step of the exchange procedure increases completion time.

Proof of Claim: If no modification is made, there is no change in completion time. So, consider a step where coveys $\{j_i, j_{u_i}\}$ and $\{j_{v_i}, j_{2k-i+1}\}$ are replaced by coveys $\{j_i, j_{2k-i+1}\}$ and $\{j_{u_i}, j_{v_i}\}$. We show the proof when $t_i + t_{2k-i+1} \leq t_{u_i} + t_{v_i}$; the proof for $t_i + t_{2k-i+1} > t_{u_i} + t_{v_i}$ is similar.

Case 1: $t_{u_i} \leq t_{v_i}$

For the following subcases, we will prove that the completion time of coveys $\{j_i, j_{u_i}\}$ and $\{j_{v_i}, j_{2k-i+1}\}$ is no more than that of coveys $\{j_i, j_{2k-i+1}\}$ and $\{j_{u_i}, j_{v_i}\}$; i.e. $\max\{T,\ t_i + t_{u_i}\} + \max\{T,\ t_{v_i} + t_{2k-i+1}\} \geq \max\{T,\ t_i + t_{2k-i+1}\} + \max\{T,\ t_{u_i} + t_{v_i}\}$.

Subcase 1.1: $t_{v_i} + t_{2k-i+1} \leq T$

$$\begin{aligned}\max\{T, t_i + t_{u_i}\} + \max\{T, t_{v_i} + t_{2k-i+1}\} &= T + T \\ &= \max\{T, t_i + t_{2k-i+1}\} + \max\{T, t_{u_i} + t_{v_i}\}\end{aligned}$$

Subcase 1.2: $t_i + t_{2k-i+1} < T \leq t_{v_i} + t_{2k-i+1}$

$$\begin{aligned}\max\{T, t_i + t_{u_i}\} + \max\{T, t_{v_i} + t_{2k-i+1}\} &= T + \max\{T, t_{v_i} + t_{2k-i+1}\} \\ &\geq T + \max\{T, t_{u_i} + t_{v_i}\} \\ &\quad (\because \max\{T, t_{v_i} + t_{2k-i+1}\} \geq \max\{T, t_{v_i} + t_{u_i}\}) \\ &= \max\{T, t_i + t_{2k-i+1}\} + \max\{T, t_{u_i} + t_{v_i}\}\end{aligned}$$

Subcase 1.3: $T \leq t_i + t_{2k-i+1}$

$$\begin{aligned}\max\{T, t_i + t_{u_i}\} + \max\{T, t_{v_i} + t_{2k-i+1}\} &= \max\{T, t_i + t_{u_i}\} + t_{v_i} + t_{2k-i+1} \\ &\geq (t_i + t_{u_i}) + (t_{v_i} + t_{2k-i+1}) \\ &= \max\{T, t_i + t_{2k-i+1}\} + \max\{T, t_{u_i} + t_{v_i}\}\end{aligned}$$

Case 2: $t_{u_i} > t_{v_i}$

Case 2 can be proved similarly to Case 1. □

When the number of machines is three, the time model is NP-complete in the strong sense.

Theorem 5. *The time model is NP-complete in the strong sense when $m = 3$.*

Proof. (By reduction from 3-PARTITION [5].) In 3-PARTITION, we are given positive integers b and $\bar{m}$ and a set $N = \{1, 2, ..., n\}$ of $n = 3\bar{m}$ elements, each having a positive integer size $a_j (< b)$ such that $\sum_{j=1}^{n} a_j = \bar{m}b$. The problem is to determine whether there exists a partition of N into $\bar{m}$ subsets, each containing exactly 3 elements from N and such

that the sum of the sizes in each subset is b. The solution is *yes* if such a partition exists, and *no* otherwise. 3-PARTITION is NP-complete in the strong sense.

Consider the instance of the time model in which $J = N$, time $t_j = a_j, j \in J$, $m = 3$, $T = b$, and $K = \bar{m}b$. Then, set J of the time model can be partitioned into disjoint subsets where the sum of elements in each subset is b if and only if 3-PARTITION has a *yes* solution. □

3.2 Unit Model

When every job has the same processing time per unit in stage 1 and the same width, but jobs might require different numbers of units to be produced, we refer to the model as the *unit model*. Consider the special case where $t_j = t \leq \frac{T}{m}$. Then,

$$\begin{aligned} C_{\max} &= \sum_i n(C_i) \times \max\Big\{ \sum_{j \in C_i} t_j,\ T \Big\} \\ &= \sum_i n(C_i) \times \max\Big\{ |C_i| \times t,\ T \Big\} \\ &= T \times \sum_i n(C_i). \end{aligned}$$

The problem description of this unit model is

Unit Model

INSTANCE: Finite set J, a time $t_j = t \leq \frac{T}{m},\ \forall j$, a production requirement $n_j \in \mathbb{Z}^+$ for each $j \in J$, a positive integer m (the number of machines), and positive numbers T (the cycle time) and K.

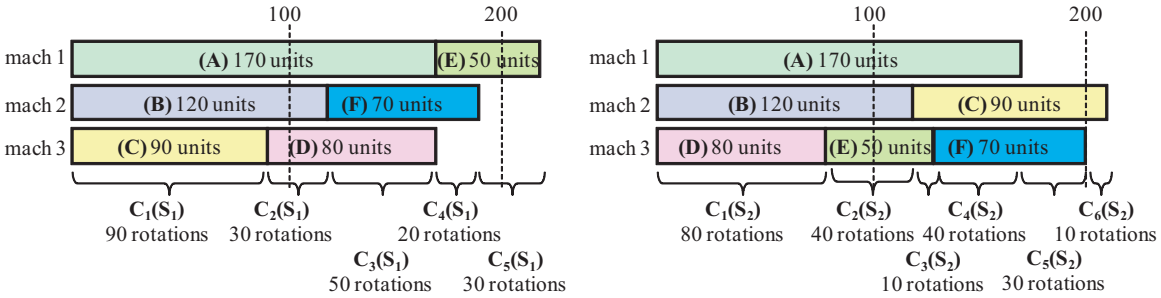
QUESTION: Does J have distinct subsets $C_1, C_2, \cdots$ with the number of rotations $n(C_i) \in \mathbb{Z}^+$ such that

$$\begin{aligned} n_j &= \sum_{k: j \in C_k} n(C_k) \\ T \times \sum_i n(C_i) &\leq K \end{aligned}$$

Coveys satisfy the restrictions **(C.1) – (C.5)**.

(as defined on the last page of Chapter II)

This unit model is identical to the parallel machine scheduling problem with objective of minimization of makespan. Since one rotation of every covey takes time equal to T, the completion time is determined by the total number of rotations of all coveys. Therefore, the completion time is $T \times \left(\sum_i n(C_i)\right)$, which corresponds to the makespan in the parallel machine scheduling problem. In addition, the restrictions of *machine dedication* and *no preemption of a product* correspond to the no preemption constraint in parallel machine scheduling.



(a) Coveys of schedule *1* have total *220* rotations. Therefore, the completion time is *220T*.

(b) Coveys of schedule *2* have total *210* rotations. Therefore, the completion time is *210T*.

Figure 3.1: Completion time of the unit model for two different schedules

In Figure 3.1, we observe the equivalence between the unit model and the parallel machine scheduling problem. Schedules 1 and 2 have different structure of coveys with six jobs, whose number of units are 170, 120, 90, 80, 70 and 50. Assume that the processing time of each job is $\frac{T}{3}$ since there are three machines. The ith covey in schedule S is represented as $C_i \in S$. The coveys of Figure 3.1 (a) total 220 rotations because $\sum_{i=1}^{5} n(C_i(S_1)) = 220$, and the coveys of Figure 3.1 (b) total 210 rotations because $\sum_{i=1}^{6} n(C_i(S_2)) = 210$. Therefore, the completion time of schedule 2 is $10 \times T$ less than that of schedule 1. If this is a parallel machine scheduling problem, the makespan of schedule 1 is 220 and that of schedule 2 is 210.

When the number of machines is at least two, the unit model is NP-complete.

Theorem 6. *The unit model is NP-complete when* $m \geq 2$.

Proof. (By reduction from MULTIPROCESSOR SCHEDULING [5].) In multiprocessor scheduling, or parallel machine scheduling, we are given set A of tasks, number $\bar{m} \in \mathbb{Z}^+$ of

processors, length $l(a) \in \mathbb{Z}^+$ for each $a \in A$, and a deadline $D \in \mathbb{Z}^+$. The problem is to determine whether there is an $\bar{m}$-processor schedule for A that meets the overall deadline D. It is NP-complete for $\bar{m} \geq 2$.

Consider an instance of the unit model with $J = A, n_j = l(a), m = \bar{m}$ and $K = D$. The completion time is $T \times \big(\sum_i n(C_i)\big)$. The value of $T \times \sum_i n(C_i) \leq K$ if and only if MULTIPROCESSOR SCHEDULING has a *yes* solution. □

3.3 Width Model

When every job requires the same processing time and the same number of units to be produced, but jobs might require different widths, we refer to the model as the *width model*. The width model is stated as follows:

Width Model

INSTANCE: Finite set J, width $w_j \in \mathbb{Z}^+$ for each $j \in J$, positive integers m and K.

QUESTION: Does J have distinct subsets $C_1, C_2, \ldots, C_k$ such that $k = \lceil \frac{|J|}{m} \rceil$ and

$$\begin{aligned} |C_i| &\leq m, \qquad \forall i \quad \text{(at most } m \text{ jobs in each covey)} \\ w(C_i) &= \max_{j \in C_h:\ \forall h \geq i} w_j, \qquad \forall i \quad \text{(monotonic ribbon width)} \\ \sum_i \Big\{ \sum_{j \in C_i} \Big(w(C_i) - w_j \Big) \Big\} &\leq K \qquad \text{(the amount of layout scrap)} \end{aligned}$$

Proposition 7. *Assume that the number of jobs in each covey is fixed; $|C_i| = K_i$ for all i where each K_i is a constant. The monotonic solution that satisfies*

$$\text{“}\textit{if } w_p > w_q \textit{ for some } p \in C_i \textit{ and } q \in C_j, \textit{ then } i \leq j\text{”}$$

gives the optimal solution for the optimization version of the width model.

Proof. Throughout this proof, we assume that the number of jobs in each covey is fixed so that $|C_i| = K_i$ for all i and each K_i is a constant. Suppose that a solution S^* satisfies the monotonic property:

$$\text{“if } w_p > w_q \text{ for some } p \in C_i \text{ and } q \in C_j, \text{ then } i \leq j.\text{”}$$

Without loss of generality, assume that jobs $j_1, \ldots, j_n$ satisfy $w_1 \geq w_2 \geq \ldots \geq w_n$. Then, the monotonic solution satisfies

$$C_i = \Big\{ j_q : \quad 1 \leq q \leq |C_1| \qquad \text{if } i = 1,$$
$$1 + \sum_{p=1}^{i-1} |C_p| \leq q \leq \sum_{p=1}^{i} |C_p| \quad \text{if } 2 \leq i \leq k \Big\}. \tag{3.1}$$

That is, in this monotonic solution, the first covey consists of the widest $|C_1|$ jobs and the second covey consists of the next widest $|C_2|$ jobs among the remaining jobs, which exclude the jobs assigned to C_1, and so on.

Now, consider a solution $S_1 \neq S^*$ with a fixed number of jobs in each covey. We can transform S_1 into S^* using the following $(k-1)$step exchange procedure where k is the number of coveys. At each step i of the procedure, for $1 \leq i \leq k-1$, if covey i already contains the widest $|C_i|$ jobs among coveys $i, \ldots, k$, i.e. C_i has the jobs as in (3.1), then move on to the next i. Otherwise, let W be a set of jobs in coveys $C_{i+1}, \ldots, C_k$ such that covey C_i would satisfy (3.1) if the $|W|$ jobs in W were swapped with the $|W|$ smallest jobs currently in covey C_i. We can do this swap one at a time, in decreasing order of width among jobs in W and among the $|W|$ smallest jobs in covey C_i.

We claim that no step of the exchange procedure increases the objective value of the optimization version of the width model. Note that the property of non-increasing ribbon width ensures that any covey C_j is run on a ribbon whose width is equal to the widest job in any of coveys $C_j, \ldots, C_k$. Now, we show that a single job swap does not increase the objective function value (i.e., the total layout scrap). Suppose job u in covey C_i is being swapped with job v in covey C_j $(i+1 \leq j \leq k)$.

First, we show that other coveys rather than C_i and C_j incur the same amount of layout scrap. (i) Coveys $C_1, \ldots, C_{i-1}$ will incur the same amount of layout scrap. The jobs in those coveys have not changed, and the ribbon width for each of those coveys also has not changed since the swap involved coveys later than C_{i-1}. (ii) Coveys C_l, for $j < l \leq k$, incur the same amount of layout scrap. The jobs in those coveys have not changed, and the ribbon width for each of those coveys also has not changed since there were no changes to coveys $l, \ldots, k$. (iii) Coveys $C_{i+1}, \ldots, C_{j-1}$ (if any) have the same jobs, but a wider job v

from covey C_j is swapped for a narrower job u from covey C_i. If v was the unique widest job that set the ribbon width for any of coveys $C_{i+1}, \ldots, C_{j-1}$, then such coveys can be run on a narrower ribbon, thereby decreasing the total layout scrap. If v was not the unique widest job, then the ribbon width is the same to those coveys, incurring the same amount of layout scrap before and after the swap.

Second, we show that coveys C_i and C_j incur no more layout scrap than before the swap. (i) Covey C_i will be run at the same width W_i as before. Thus, the ribbon width of Covey C_i has not changed. Therefore, the jobs in covey C_i not involved in the swap will incur the same layout loss. (ii) Let W_j be the width that covey C_j was run at before the swap, and let W_j' be the width that covey C_j is run at after the swap. Note that since $w_v > w_u$, $W_j \geq W_j'$. So, the jobs in covey C_j not involved in the swap will incur no more layout loss than before. (iii) Job v incurs $(W_i - W_j)$ extra layout scrap. Job u incurs $(W_i - W_j')$ less layout scrap. Since $(W_i - W_j') \geq (W_i - W_j)$, jobs u and v together incur no more layout scrap than before. □

The following two-stage approach solves the width model. In the first stage, we consider all possible distributions of coveys; we determine how many jobs exist in each covey. In the second stage, for each distribution in which the number of jobs in each covey is fixed, we find the best assignment of jobs to coveys in order to minimize layout scrap. By this two-stage approach, we look through full enumerations of solutions and we can find an optimal solution. From Proposition 7, we already know that the monotonic solution gives an optimal to each of the second stage problems. Therefore, once we start with the monotonic solution, we need to execute only the first stage. The following dynamic programming gives an optimal solution to the first stage problem and thus gives an overall optimal solution to the width model.

Dynamic Programming Approach

Using the above monotonic property, the following dynamic programming solves the width model. Assume that $w_{j_1} \geq w_{j_2} \geq \ldots \geq w_{j_{|J|}}$ and jobs are assigned to the earlier coveys in the

order of $j_1, j_2, \ldots, j_{|J|}$ by the monotonic property. The stage of our dynamic programming for the width model is the number of coveys, denoted by k. The state space is (p, q) where p is the number of jobs assigned to coveys $C_1, C_2, \ldots, C_k$ and q is the number of jobs in the last covey, C_k. By the definition of the width model, we know that $(k-1)m + 1 \leq p \leq km$ and $p - (k-1)m \leq q \leq m$ where m is the number of machines. We denote layout scrap of covey C as $LS(C)$ and it is defined as

$$LS(C) = \sum_{j \in C} \Big(\max_{i \in C} w_i - w_j \Big).$$

Let $f_k(p, q)$ be the least possible layout scrap in coveys $C_1, \ldots, C_k$ where p number of jobs are assigned and the last covey has q number of jobs. When $k = 1$, we initialize $f_1(p, q) = f_1(p, p) = LS(\{j_1, j_2, \ldots, j_p\})$. When $k \geq 2$, we have the following recursion:

$$\begin{aligned} f_k(p, q) &= \min_{(p-q)-(k-2)m \leq r \leq m} f_{k-1}(p - q, r) + LS(C_k) \\ &= \min_{(p-q)-(k-2)m \leq r \leq m} f_{k-1}(p - q, r) + LS(\{j_{p-q+1}, \ldots, j_p\}). \end{aligned}$$

In words, this recursion means the following: when p number of jobs are assigned and the last covey has q number of jobs, the least possible layout scrap in coveys $C_1, \ldots, C_k$ consists of two parts. The first part is the minimum value of the previous stage and we exclude q number of jobs assigned to the last covey C_k. Hence, in the previous stage, $(p - q)$ number of jobs are assigned to $C_1, C_2, \ldots, C_{k-1}$. Among these alternatives, we take the minimum value. The second part is the layout scrap of the last covey, C_k to which q number of jobs are assigned. Therefore, we have the following theorem.

Theorem 8. *A dynamic programming gives an optimal solution to the width model in polynomial time $O(nm)$ where n is the number of jobs and m is the number of machines.*

Proof. The above forward recursion of dynamic programming gives an optimal solution. An upper bound of the state space of (p, q) is that p can be the number of jobs n and for each p there exist at most m jobs are assigned to the last covey. Therefore, the dynamic programming runs in polynomial time $O(nm)$. □

The following example illustrates the dynamic programming algorithm.

Example 9. *Widths of a set* $J = \{12, 10, 9, 8, 8, 7, 6, 4, 4, 3\}$, $m = 3$

Stage 1 : $k = 1,\ 1 \le p \le 3$

$f_1(1,1) = LS(\{12\}) = 0 \qquad \blacktriangleright C_1 = \{12\}$

$f_1(2,2) = LS(\{12,10\}) = 2 \qquad \blacktriangleright C_1 = \{12,10\}$

$f_1(3,3) = LS(\{12,10,9\}) = 5 \qquad \blacktriangleright C_1 = \{12,10,9\}$

Stage 2 : $k = 2,\ 4 \le p \le 6$

$f_2(4,1) = \min f_1(3,-) + LS(\{8\}) = 5 + 0 = 5 \qquad \blacktriangleright C_1 = \{12,10,9\}, C_2 = \{8\}$

$f_2(4,2) = \min f_1(2,-) + LS(\{9,8\}) = 2 + 1 = 3 \qquad \blacktriangleright C_1 = \{12,10\}, C_2 = \{9,8\}$

$f_2(4,3) = \min f_1(1,-) + LS(\{10,9,8\}) = 0 + 3 = 3 \qquad \blacktriangleright C_1 = \{12\}, C_2 = \{10,9,8\}$

$f_2(5,2) = \min f_1(3,-) + LS(\{8,8\}) = 5 + 0 = 5 \qquad \blacktriangleright C_1 = \{12,10,9\}, C_2 = \{8,8\}$

$f_2(5,3) = \min f_1(2,-) + LS(\{9,8,8\}) = 2 + 2 = 4 \qquad \blacktriangleright C_1 = \{12,10\}, C_2 = \{9,8,8\}$

$f_2(6,3) = \min f_1(3,-) + LS(\{8,8,7\}) = 5 + 1 = 6 \qquad \blacktriangleright C_1 = \{12,10,9\}, C_2 = \{8,8,7\}$

Stage 3 : $k = 3,\ 7 \le p \le 9$

$f_3(7,1) = \min f_2(6,-) + LS(\{6\}) = f_2(6,3) + 0 = 6 + 0 = 6$

$\blacktriangleright C_1 = \{12,10,9\}, C_2 = \{8,8,7\}, C_3 = \{6\}$

$f_3(7,2) = \min f_2(5,-) + LS(\{7,6\}) = f_2(5,3) + 1 = 4 + 1 = 5$

$\blacktriangleright C_1 = \{12,10\}, C_2 = \{9,8,8\}, C_3 = \{7,6\}$

$f_3(7,3) = \min f_2(4,-) + LS(\{8,7,6\}) = f_2(4,2) + 3 = 3 + 3 = 6$

$\blacktriangleright C_1 = \{12,10\}, C_2 = \{9,8\}, C_3 = \{8,7,6\}$

$f_3(8,2) = \min f_2(6,-) + LS(\{6,4\}) = f_2(6,3) + 1 = 6 + 1 = 7$

$\blacktriangleright C_1 = \{12,10,9\}, C_2 = \{8,8,7\}, C_3 = \{6,4\}$

$f_3(8,3) = \min f_2(5,-) + LS(\{7,6,4\}) = f_2(5,3) + 4 = 4 + 4 = 8$

$\blacktriangleright C_1 = \{12,10\}, C_2 = \{9,8,8\}, C_3 = \{7,6,4\}$

$$
\begin{aligned}
f_3(9,3) &= \min f_2(6,-) + LS(\{6,4,4\}) = f_2(6,3) + 4 = 6 + 4 = 10 \\
&\quad \blacktriangleright C_1 = \{12,10,9\}, C_2 = \{8,8,7\}, C_3 = \{6,4,4\}
\end{aligned}
$$

$$
\begin{aligned}
\textit{Stage 4} &: \quad k = 4,\ p = 10 \\
f_4(10,1) &= \min f_3(9,-) + LS(\{3\}) = f_3(9,3) + 0 = 10 + 0 = 10 \\
&\quad \blacktriangleright C_1 = \{12,10,9\}, C_2 = \{8,8,7\}, C_3 = \{6,4,4\}, C_4 = \{3\} \\
f_4(10,2) &= \min f_3(8,-) + LS(\{4,3\}) = f_3(8,2) + 1 = 7 + 1 = 8 \\
&\quad \blacktriangleright C_1 = \{12,10,9\}, C_2 = \{8,8,7\}, C_3 = \{6,4\}, C_4 = \{4,3\} \\
f_4(10,3) &= \min f_3(7,-) + LS(\{4,4,3\}) = f_3(7,2) + 1 = 5 + 1 = 6 \\
&\quad \blacktriangleright C_1 = \{12,10\}, C_2 = \{9,8,8\}, C_3 = \{7,6\}, C_4 = \{4,4,3\} \qquad \leftarrow \textit{optimum}
\end{aligned}
$$

3.4 Time and Unit Model

If every job has the same width, but jobs require different processing times per unit in stage 1 and different numbers of units to be produced, then we refer to the remaining model as the *time and unit model*. Since the special case of the unit model, is NP-complete for $m \geq 2$, the time and unit model is NP-complete when $m \geq 2$, and it is stated as follows:

Time and Unit Model

INSTANCE: Finite set J, a processing time $t_j \in \mathbb{R}^+$ and a production requirement $n_j \in \mathbb{Z}^+$ for each $j \in J$, a positive integer m (the number of machines), and positive numbers T (the cycle time) and K.

QUESTION: Does J have distinct subsets $C_1, C_2, \cdots$ with the number of rotations $n(C_i) \in \mathbb{Z}^+$ such that

$$
\begin{aligned}
n_j &= \sum_{i:j \in C_i} n(C_i) \\
\sum_i n(C_i) \times \max\Big\{ \sum_{j \in C_i} t_j,\ T \Big\} &\leq K
\end{aligned}
$$

Coveys satisfy the restrictions **(C.1)** – **(C.5)**.

(as defined on the last page of Chapter II)

In Chapter IV, we propose a heuristic algorithm for this time and unit model and analyze its worst case performance.

3.5 Time and Width Model

When every job requires the same number of units to be produced, but jobs might require different processing times and widths, we refer to the model as the *time and width model*. The time and width model is stated as follows:

Time and Width Model

INSTANCE: Finite set J, a time $t_j \in \mathbb{R}^+$, width $w_j \in \mathbb{Z}^+$ for each $j \in J$, positive integers m, positive numbers T and K.

QUESTION: Does J have distinct subsets $C_1, C_2, \cdots C_k$ such that $k = \lceil \frac{|J|}{m} \rceil$ and

$$\begin{aligned} |C_i| &\leq m, \qquad \forall i \qquad \text{(at most } m \text{ jobs in each covey)} \\ w(C_i) &= \max_{j \in C_h:\ \forall h \geq i} w_j, \qquad \forall i \qquad \text{(monotonic ribbon width)} \\ K &\geq \sum_i \max\Big\{ \sum_{j \in C_i} \frac{w(C_i)}{w_j} t_j,\ T \Big\} \qquad \text{(completion time)} \end{aligned}$$

For $m = 2$, the following algorithm solves the optimization version of the time and width model.

Algorithm: An algorithm using minimum weighted matching

(Step 1) Construct a complete graph in which a node represents a job. If the number of jobs is odd, add one dummy job j of zero time $t_j = 0$ and $w_j = 1$. The edge between nodes a and b has the weight of

$$\max\Big\{ \frac{\max\{w_a, w_b\}}{w_a} t_a + \frac{\max\{w_a, w_b\}}{w_b} t_b,\ T \Big\}.$$

This edge represents a covey consisting of jobs a and b. The edge weight represents the completion time of this covey. The width of this covey is defined as $\max\{w_a, w_b\}$.

(Step 2) Apply minimum weighted matching algorithm.

(Step 3) Sort matched sets by non-increasing order in widths of coveys. Then, we assign these sorted sets to coveys $C_1, C_2, \ldots, C_k$

Theorem 10. *An algorithm using* minimum weighted matching *produces an optimal solution for the optimization version of the time and width model when the number of machines is two.*

Proof. When the number of jobs is even, each covey must have two jobs because the number of coveys is $k = \lceil \frac{|J|}{m} \rceil$. Thus, a covey consists of any combination of two jobs, and each covey is represented as an edge between two nodes (two jobs) on a complete graph. It follows that any solution of perfect matching on this complete graph corresponds to a feasible solution to the time and width model and all alternatives of perfect matching on this complete graph correspond the full enumeration of feasible solutions for the time and width model. Therefore, finding minimum weighted matching on this complete graph is equivalent to finding an optimal solution for the optimization version of the time and width model. Note that by Step 3, we do not have extra scrap due to the property of the monotonic ribbon width. When the number of jobs is odd, only one covey consists of one job and each of other $(k-1)$ coveys consists of two jobs. By adding a dummy job j of $t_j = 0$ and $w_j = 1$, there is no contribution to the sum of snap times and no layout scrap is incurred in the covey that has this dummy job. Since the complexity of minimum weighted matching is $O(|V|^2\ |E|)$, the complexity of this algorithm is $O(n^4)$ because $|V| = n$ and $|E| = \frac{n(n-1)}{2}$. □

Since the time model is NP-complete for $m = 3$, the time and width model is also NP-complete when $m = 3$.

3.6 Full Model: Time, Unit, and Width Model

The full model considers all three elements together: stage 1 processing times, widths and required production units for each job. Since the special case of the unit model is NP-complete for $m \geq 2$, the full model is also NP-complete when $m \geq 2$, as follows:

Full Model: Time, Unit and Width Model

INSTANCE: Finite set J, a processing time $t_j \in \mathbb{R}^+$, a production requirement $n_j \in \mathbb{Z}^+$, and width $w_j \in \mathbb{Z}^+$ for each $j \in J$, a positive integer m (the number of machines), and positive numbers T (the cycle time) and K.

QUESTION: Does J have distinct subsets $C_1, C_2, \cdots$ with the number of rotations $n(C_i) \in \mathbb{Z}^+$ such that

$$\begin{aligned} n_j &= \sum_{i: j \in C_i} n(C_i) \\ w(C_i) &= \max_{j \in C_h:\ \forall h \geq i} w_j \\ \sum_i n(C_i) \times \max\Big\{ \sum_{j \in C_i} \frac{w(C_i)}{w_j} t_j,\ T \Big\} &\leq K \end{aligned}$$

Coveys satisfy the restrictions **(C.1)** – **(C.5)**.

(as defined on the last page of Chapter II)

CHAPTER IV

A HEURISTIC AND PERFORMANCE ANALYSIS OF FGSP

In the previous chapter, we analyzed FGSP by considering simple sub-models. We also proved that FGSP is NP-hard. Moreover, for the size of the problem, that need to be solved in practice, methods that are capable of proving optimality are not practical. In this chapter, we propose a simple heuristic algorithm, called the *Longest Unit First (LUF)*, and analyze its worst case performance for the time and unit model. The *Longest Processing Time First (LPT)* algorithm (Graham [9]) for parallel machine scheduling motivates the proposed LUF algorithm.

Algorithm: Longest Unit First (LUF)

(Step 0) Choose m jobs with the m largest number of units. The first covey consists of these m jobs. For $i = 1, \ldots, m$, the first-stage machine produces a unit of the ith job in the covey, and it is assigned to the ith machine at the second-stage.

(Step 1) The procedure of producing a unit of each job in the covey is repeated until all of the units of a job in the covey are completed.

(Step 2) The completed job exits from the covey, and a new job with the largest number of units among unassigned jobs enters into the covey, forming a new covey. This entered job is assigned to the second-stage machine to which the exited job was assigned.

(Step 3) Steps 1 and 2 are repeated until all jobs are done.

For example, suppose we have three machines and six jobs: job A requires 170 units, B 120, C 90, D 80, E 50, and F 70 units. First, the three jobs with the most required production, jobs A, B, and C, are assigned to machines. Job A is assigned to machine 1, then job B is assigned to machine 2, and job C is assigned to machine 3. This is the first covey. At this point, all machines are busy for 90 rotations, until job C finishes on machine

3. Job D, the most required production among unassigned jobs, is then assigned to machine 3. Then, job F is assigned to machine 2 after another 30 rotations of coveys (120 total), and job E can be assigned to either machine 1 or machine 3 after 50 more rotations (170 total). The schedule produced by the LUF algorithm is shown in Figure 3.1 (a).

To analyze the worst case performance of LUF, we need to distinguish between the schedule produced by LUF and the optimal schedule. We denote by schedules $S, \hat{S}$ and S^* the following:

S : an arbitrary schedule

$\hat{S}$: a schedule produced by the *Longest Unit First* algorithm

S^* : an optimal schedule.

The sum of rotations for the coveys of schedule S is defined as

$$N(S) := \sum_{i:C_i \in S} n(C_i).$$

Recall that a covey in schedule S is $C_i \in S$.

In the following section, we first present notation and some basic properties that will be used in the worst case analysis. Then, we will prove the following theorem for two cases according to $N(S), N(\hat{S})$ and $N(S^*)$.

Theorem 11. *For any schedule S with $N(S) \leq N(S^*)$, we have*

$$C_{\max}(S) \leq \Big(1 + \frac{m-1}{m}\Big) C_{\max}(S^*),$$

where m is the number of second-stage machines. For schedule $\hat{S}$ with $N(\hat{S}) > N(S^*)$, produced by the LUF *algorithm, we have*

$$C_{\max}(\hat{S}) \leq \Big(\big(1 + \frac{m-1}{m}\big) \; + \; \big(\frac{1}{3} - \frac{1}{3m}\big) \Big) C_{\max}(S^*).$$

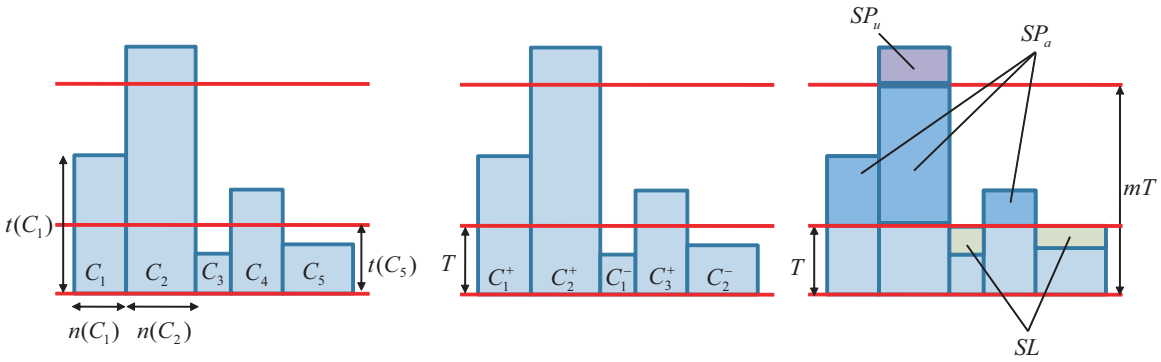
From the first result of Theorem 11, if we minimize the number of rotations of schedule S so that $N(S) \leq N(S^*)$, a worst case bound for FGSP is $\Big(1 + \frac{m-1}{m}\Big)$. This bound is obtained by only considering the unit element of the problem, i.e., minimizing the makespan, which is known to be NP-hard.

4.1 Basic Properties

The following notation will be used in the proof. In Figure 4.1, the horizontal-axis represents the number of rotations of each covey and the vertical-axis represents the sum of times of jobs in a covey. From Figure 4.1 (a), we can identify the number of rotations, $n(C_i)$ and the sum of times, $t(C_i)$ for each covey C_i.

To calculate the completion time, we analyze the sum of stage 1 machine times $t(C_i)$ and the number of rotations required $n(C_i)$ for each covey C_i of the schedule. We partition coveys $C_1, C_2, \ldots, C_K$ of a schedule S into two sets: a set C^- of slack coveys that incur scrap at each rotation because their total stage 1 processing time is less than T and a set C^+ of surplus coveys whose total stage 1 processing time is at least T per rotation. They are defined as

$$C^- := \{C_i \in S : \sum_{j \in C_i} t_j < T\} \qquad \text{and} \qquad C^+ := \{C_i \in S : \sum_{j \in C_i} t_j \geq T\}.$$



(a) sum of times and the number of rotations for coveys

(b) slack and surplus coveys

(c) slack time, unavoidable surplus time and avoidable surplus time

Figure 4.1: Slack coveys and surplus coveys

We denote slack coveys as $C_1^-, C_2^-, \ldots, C_{|C^-|}^-$ and denote surplus coveys as $C_1^+, C_2^+, \ldots, C_{|C^+|}^+$, as illustrated in Figure 4.1 (b). In addition, we denote the sum of rotations for all coveys in schedule S as $N(S)$, the sum of rotations for slack coveys as $N^-(S)$, and the sum of rotations for surplus coveys as $N^+(S)$. In other words,

$$N(S) := \sum_{C_i \in S} n(C_i), \quad N^-(S) := \sum_{C_i^- \in C^-} n(C_i^-), \quad \text{and} \quad N^+(S) := \sum_{C_i^+ \in C^+} n(C_i^+).$$

Clearly, $N(S) = N^-(S) + N^+(S)$.

Consider the following two lower bounds for the optimal solution to this problem.

$$\textbf{Lower Bound 1:} \qquad C_{\max}(S^*) \geq \sum_{j=1}^{J} n_j \; t_j$$

$$\textbf{Lower Bound 2:} \qquad C_{\max}(S^*) \geq N(S^*) \; T + \sum_{j=1}^{J} n_j \; \max(t_j - T, \; 0)$$

The first lower bound is time-based; the completion time is trivially no less than the sum of all stage 1 processing times. The second trivial lower bound, based on units, is that $C_{\max}(S^*) \geq N(S^*) \; T$ since each rotation must take at least T time including slack if necessary. However, we can tighten this bound. If a single job j requires more stage 1 processing time than T, then each rotation of its covey will have at least $t_j - T$ surplus time. Lower Bound 2 includes this amount of surplus time that cannot be eliminated.

In our analysis, we need to calculate the total amount of slack time, the total surplus time up to $(m-1)T$ per rotation and the total surplus time above $(m-1)T$ per rotation in schedule S. For a schedule S, we define $SL(S), SP_a(S)$, and $SP_u(S)$ as the following:

$$\begin{aligned}
SL(S) &:= \sum_{C_i^- \in C^-} n(C_i^-) \; \Big(T - t(C_i^-)\Big) \\
SP_a(S) &:= \sum_{C_i^+ \in C^+} n(C_i^+) \; \min\Big\{(m-1)T, \; t(C_i^+) - T\Big\} \\
SP_u(S) &:= \sum_{C_i^+ \in C^+} n(C_i^+) \; \max\Big\{0, \; t(C_i^+) - mT\Big\}
\end{aligned}$$

Note that SP_a might be reduced and $C_{\max}$ can be decreased by increasing slack time SL. Hence, we call SP_a *avoidable surplus time*. Meanwhile, SP_u is no more than $\sum_{j=1}^{J} n_j \; \max(t_j - T, \; 0)$. Therefore, SP_u cannot be reduced, and we call SP_u *unavoidable surplus time*. In Figure 4.1 (c), we identify $SL(S), SP_a(S)$, and $SP_u(S)$.

The completion time of a schedule S is $\big\{$ (number of rotations)$\times$ (cycle time) plus (amount of surplus time) $\big\}$. That is,

$$C_{\max}(S) = N(S) \times T + SP_a(S) + SP_u(S). \tag{4.1}$$

Therefore, roughly speaking, in FGSP there are two ways to minimize the completion time of schedule S, related to the main elements of *time* and *units* that are discussed individually

in Sections 3.1 and 3.2. In the aspect of *units*, the rotations of the schedule $N(S)$, can be reduced. In the aspect of *time*, the amount of surplus time, i.e. only $SP_a(S)$, can be reduced.

The above lower bounds can be represented in terms of slack time and surplus time in the following lemmas.

Lemma 12. *For an arbitrary schedule S,*

$$N(S)\ T + SP_a(S) + SP_u(S) - SL(S) \leq C_{\max}(S^*). \tag{4.2}$$

Proof. By definition, the sum of stage 1 processing times for all jobs is equivalent to $N(S)\ T$ +(total surplus time) − (total slack time) for any schedule S. Therefore, we have

$$\begin{aligned} C_{\max}(S^*) &\geq \sum_{j=1}^{J} n_j\ t_j \qquad (\because \text{Lower Bound 1}) \\ &= N(S)\ T + SP_a(S) + SP_u(S) - SL(S). \end{aligned}$$

□

Lemma 13. *For any schedule S with $N(S) \leq N(S^*)$,*

$$N(S)\ T + SP_u(S) \leq C_{\max}(S^*).$$

Proof.

$$\begin{aligned} C_{\max}(S^*) &\geq N(S^*)\ T + \sum_{j=1}^{J} n_j\ \max(t_j - T,\ 0) \qquad (\because \text{Lower Bound 2}) \\ &\geq N(S)\ T + SP_u(S) \\ &\qquad (\because N(S) \leq N(S^*) \text{ and } SP_u(S) \leq \sum_{j=1}^{J} n_j\ \max(t_j - T,\ 0)) \end{aligned}$$

□

4.2 Worst Case Bound for Any Schedule S with $N(S) \leq N(S^*)$

In this section, we formalize a worst case bound on the cost of any schedule that has no more rotations than the optimal schedule.

First, we consider some basic properties of the amount of slack time and surplus time that will be useful in the analysis. By the definition of slack time, we have

$$SL(S) \leq N^{-}(S)\ T \tag{4.3}$$

because $SL(S) = \sum_{C_i^- \in C^-} n(C_i^-)\big(T - t(C_i^-)\big) \leq \sum_{C_i^- \in C^-} n(C_i^-)T = N^-(S)T$. Similarly, by the definition of surplus time, we have

$$SP_a(S) \leq (m-1)\ N^{+}(S)\ T \tag{4.4}$$

because $SP_a(S) = \sum_{C_i^+ \in C^+} n(C_i^+) \min\big\{(m-1)T,\ t(C_i^+) - T\big\} \leq \sum_{C_i^+ \in C^+} n(C_i^+)(m-1)T = (m-1)N^+(S)T$.

Lemma 14. *For any schedule S with $N(S) \leq N(S^*)$,*

$$C_{\max}(S) \leq \Big(1 + \frac{m-1}{m}\Big) C_{\max}(S^*).$$

Proof. We consider two cases: $SL(S) > SP_a(S)$ and $SL(S) \leq SP_a(S)$.

Case (i): $SL(S) > SP_a(S)$

By Lemma 13, we have

$$N(S)\ T + SP_u(S) \leq C_{\max}(S^*). \tag{4.5}$$

Since $SL(S) \leq N^-(S)\ T$ (inequality (4.3)) and $SL(S) > SP_a(S)$, we have $(m-1)\ SP_a(S) < (m-1)\ SL(S) \leq (m-1)\ N^-(S)\ T$. By inequality (4.4), $SP_a(S) \leq (m-1)\ N^+(S)\ T$. Therefore, the sum of these two inequalities yields

$$m\ SP_a(S) < (m-1)\ \big(N^-(S) + N^+(S)\big)\ T = (m-1)\ N(S)\ T.$$

Because we know that $N(S)\ T \leq N(S^*)\ T \leq C_{\max}(S^*)$, we have

$$SP_a(S) < \frac{m-1}{m}\ C_{\max}(S^*). \tag{4.6}$$

By inequalities (4.5) and (4.6), we have

$$N(S)\ T + SP_a(S) + SP_u(S) < \Big(1 + \frac{m-1}{m}\Big) C_{\max}(S^*).$$

Because $N(S)\ T + SP_a(S) + SP_u(S) = C_{\max}(S)$, we finally have

$$C_{\max}(S) < \Big(1 + \frac{m-1}{m}\Big) C_{\max}(S^*).$$

Case (ii): $SL(S) \leq SP_a(S)$

By Lemma 12, we have

$$N(S)\ T + SP_a(S) + SP_u(S) - SL(S) \leq C_{\max}(S^*). \tag{4.7}$$

By inequality (4.3), we have $(m-1)\ SL(S) \leq (m-1)\ N^-(S)\ T$. Since $SP_a(S) \leq (m-1)\ N^+(S)\ T$ (inequality (4.4)) and $SL(S) \leq SP_a(S)$, we have $SL(S) \leq SP_a(S) \leq (m-1)\ N^+(S)\ T$. Therefore, the sum of these two inequalities yields

$$\begin{aligned} m\ SL(S) &\leq (m-1)\ \big(N^-(S) + N^+(S)\big)\ T \\ &= (m-1)\ N(S)\ T \\ &\leq (m-1)\ N(S^*)\ T \qquad (\because N(S) \leq N(S^*)) \\ &\leq (m-1)\ C_{\max}(S^*). \end{aligned}$$

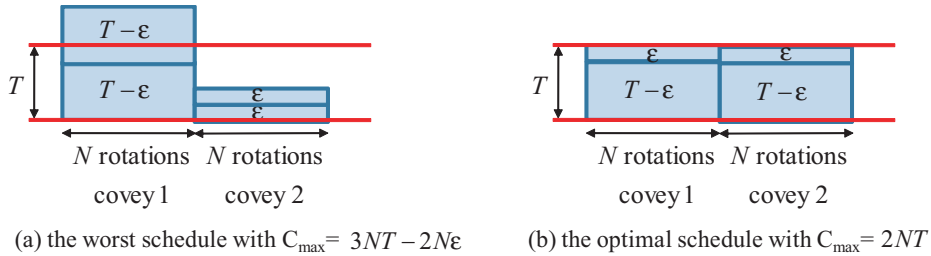
Therefore, we have

$$SL(S) \leq \frac{m-1}{m}\ C_{\max}(S^*). \tag{4.8}$$

By inequalities (4.7) and (4.8), we have

$$C_{\max}(S) = N(S)\ T + SP_a(S) + SP_u(S) \leq \Big(1 + \frac{m-1}{m}\Big) C_{\max}(S^*).$$

□



(a) the worst schedule with $C_{max} = 3NT - 2N\varepsilon$ (b) the optimal schedule with $C_{max} = 2NT$

Figure 4.2: A worst case instance when m=2 and $N(S) \leq N(S^*)$

Now, we present a tight instance of the worst case bound. When the number of machines is two, we have

$$C_{\max}(S) \leq \frac{3}{2}\, C_{\max}(S^*).$$

An instance of the tight worst bound for $m = 2$ is illustrated in Figure 4.2. Jobs A and B have $(T - \epsilon)$ processing time with N units of each required and jobs C and D have ϵ processing time with N units required. In Figure 4.2 (a) we show the following schedule S_a: the first covey consists of jobs A and B for N rotations and the second covey consists of jobs C and D for N rotations. The completion time of schedule S_a is

$$\begin{aligned} C_{\max}(S_a) &= N\ \max\{2(T-\epsilon),\ T\} + N\ \max\{2\epsilon,\ T\} \\ &= 2N\ (T-\epsilon) + N\ T \\ &= 3NT - 2N\epsilon. \end{aligned}$$

The schedule S_b illustrated in Figure 4.2 (b) has jobs A and C in the first covey and jobs B and D in the second covey. The completion time of schedule S_b is

$$\begin{aligned} C_{\max}(S_b) &= N\ \max\{(T-\epsilon)+\epsilon,\ T\} + N\ \max\{(T-\epsilon)+\epsilon,\ T\} \\ &= 2NT. \end{aligned}$$

Therefore, if ϵ goes to zero, $\frac{C_{\max}(S_a)}{C_{\max}(S_b)} = \frac{3}{2}$, showing that the worst case bound is tight. (It also implies that S_b is an optimal schedule and S_a is a worst-possible schedule.)

4.3 Worst Case Bound of the Longest Unit First *algorithm*

In this section, we use results from the previous section to prove a worst case performance bound for the *LUF* algorithm. If schedule $\hat{S}$ produced by the *LUF* algorithm satisfies $N(\hat{S}) \leq N(S^*)$, the worst case bound of Lemma 14 holds for this schedule. Now, consider the opposite case, $N(\hat{S}) > N(S^*)$; when the number of rotations of schedule $\hat{S}$ is greater than that of the optimal schedule.

We partition the rotations of $\hat{S}$ into two subschedules, $\hat{S}_1$ and $\hat{S}_2$, such that $\hat{S}_1$ contains the $N(S^*)$ rotations of $\hat{S}$ with the longest stage 1 processing times, and $\hat{S}_2$ contains the $N(\hat{S}) - N(S^*)$ rotations with the shortest stage 1 processing times.

We define slack time and surplus time for $\hat{S}_1$ and $\hat{S}_2$ as before. Then, the inequalities (4.3) and (4.4) still hold for $\hat{S}_1$ and $\hat{S}_2$. Since $\hat{S}_1$ is a subset of $\hat{S}$, we know that the sum of processing times of jobs in schedule $\hat{S}_1$ is no more than that in schedule $\hat{S}$, which means that

$$\begin{aligned} & N(\hat{S}_1)\ T + SP_a(\hat{S}_1) + SP_u(\hat{S}_1) - SL(\hat{S}_1) \\ & \leq N(\hat{S})\ T + SP_a(\hat{S}) + SP_u(\hat{S}) - SL(\hat{S}). \end{aligned} \tag{4.9}$$

Therefore, Lemma 12 still holds:

$$N(\hat{S}_1)\ T + SP_a(\hat{S}_1) + SP_u(\hat{S}_1) - SL(\hat{S}_1) \ \leq \ C_{\max}(S^*).$$

Similar to the proof of Lemma 13, we have

$$\begin{aligned} \sum_{j=1}^{J} n_j\ \max(t_j - T,\ 0) \ &\geq \ SP_u(\hat{S}) \\ &\geq \ SP_u(\hat{S}_1). \end{aligned}$$

In addition, we have

$$\begin{aligned} C_{\max}(S^*) \ &\geq \ N(S^*)\ T + \sum_{j=1}^{J} n_j\ \max(t_j - T,\ 0) && (\because \text{Lower Bound 2}) \\ &= \ N(\hat{S}_1)\ T + \sum_{j=1}^{J} n_j\ \max(t_j - T,\ 0) && (\because N(\hat{S}_1) = N(S^*)) \\ &\geq \ N(\hat{S}_1)\ T + SP_u(\hat{S}_1). \end{aligned}$$

Therefore, we have a result similar with Lemma 13 for $\hat{S}_1$

$$C_{\max}(S^*) \ \geq \ N(\hat{S}_1)\ T + SP_u(\hat{S}_1).$$

Applying similar arguments do in the proof of Lemma 14, we have

$$C_{\max}(\hat{S}_1) \leq \Big(1 + \frac{m-1}{m}\Big) C_{\max}(S^*). \tag{4.10}$$

Inequality (4.10) will be used in the following proof, in which we also consider the remaining rotations $\hat{S}_2$.

Theorem 15. *A schedule $\hat{S}$ produced by the* Longest Unit First *algorithm has a worst case bound:*

$$C_{\max}(\hat{S}) \leq \Big(\big(1 + \frac{m-1}{m}\big) \; + \; \big(\frac{1}{3} - \frac{1}{3m}\big) \Big) C_{\max}(S^*). \tag{4.11}$$

Proof. When the number of machines is two, if the number of units of one job is no less than the sum of the number of units of all the other jobs, *LUF* finds a trivial optimal solution with the long job on one machine and all other jobs on the other machine. See Lemma 16 in Appendix A. The proof below excludes this trivial case.

Schedule $\hat{S}$ with $N(\hat{S}) \leq N(S^*)$ produced by *LUF* has the worst case bound of Lemma 14, which satisfies inequality (4.11). Now, consider schedule $\hat{S}$ with $N(\hat{S}) > N(S^*)$, and define $\hat{S}_1$ and $\hat{S}_2$ as above.

In the parallel machine scheduling problem, the *Longest Processing Time First (LPT)* algorithm has the following worst case bound (Graham [9]):

$$C_{\max}(LPT) \leq \Big(\frac{4}{3} - \frac{1}{3m} \Big) \, C_{\max}(OPT)$$

where *LPT* and *OPT* represent the schedule produced by the *LPT* algorithm and the schedule of an optimal solution, respectively. The *Longest Unit First (LUF)* algorithm of FGSP is similar to *LPT* and, we can apply the Graham theorem to the number of rotations for $\hat{S}_1, \hat{S}_2$ and S^* of *LUF*. We already observed that the unit model is equivalent to the parallel machine scheduling problem. By their relation, the makespan of *LPT* in the parallel machine scheduling problem, $C_{\max}(LPT)$, corresponds to the number of rotations $N(\hat{S})$ for schedule $\hat{S}$, and $C_{\max}(OPT)$ corresponds to $N(S^*)$. Hence, we have

$$\begin{aligned} N(\hat{S}_1) + N(\hat{S}_2) &\leq \Big(\frac{4}{3} - \frac{1}{3m}\Big) \, N(S^*) \\ N(\hat{S}_2) &\leq \Big(\frac{1}{3} - \frac{1}{3m}\Big) \, N(S^*) \qquad (\because N(\hat{S}_1) = N(S^*)). \end{aligned} \tag{4.12}$$

Case (i): If the sum of times for each rotation of $\hat{S}_2$ is no more than T, we have

$$\begin{aligned} C_{\max}(\hat{S}_2) &= N(\hat{S}_2) \; T \\ &\leq \big(\frac{1}{3} - \frac{1}{3m}\big) N(S^*) \; T \qquad (\because \text{inequality (4.12) }) \\ &\leq \big(\frac{1}{3} - \frac{1}{3m}\big) C_{\max}(S^*). \end{aligned}$$

Therefore, we have

$$\begin{aligned} C_{\max}(\hat{S}) = C_{\max}(\hat{S}_1) + C_{\max}(\hat{S}_2) &\leq \big(1 + \frac{m-1}{m}\big) C_{\max}(S^*) \ + \ \big(\frac{1}{3} - \frac{1}{3m}\big) C_{\max}(S^*) \\ &= \Big(\big(1 + \frac{m-1}{m}\big) \ + \ \big(\frac{1}{3} - \frac{1}{3m}\big) \Big) C_{\max}(S^*). \end{aligned}$$

Case (ii): Suppose that at least one rotation in $\hat{S}_2$ has total stage 1 processing time greater than T. Let $t_{\max}$ be the maximum sum of times among rotations in schedule $\hat{S}_2$, that is, $t_{\max} := \max_{C_i \in \hat{S}_2} t(C_i)$. Since we take $\hat{S}_2$ from $\hat{S}$ so that the sum of times for each covey in $\hat{S}_2$ is as small as possible, the total stage 1 processing time of coveys with a rotation in $\hat{S}_1$ should be no less than $t_{\max}$. Therefore, we have

$$\begin{aligned} C_{\max}(\hat{S}_2) &\leq N(\hat{S}_2)\ t_{\max} && (\because t(C_k) \leq t_{\max},\ \forall C_k \in \hat{S}_2\ ) \\ &\leq \big(\frac{1}{3} - \frac{1}{3m}\big)\ N(\hat{S}_1)\ t_{\max} && (\because \text{inequality (4.12)}\ ) \\ &\leq \big(\frac{1}{3} - \frac{1}{3m}\big) \sum_{C_k \in \hat{S}_1} n(C_k)\ t(C_k) && (\because t(C_k) \geq t_{\max},\ \forall C_k \in \hat{S}_1\ ) \\ &\leq \big(\frac{1}{3} - \frac{1}{3m}\big) \sum_{C_k \in \hat{S}} n(C_k)\ t(C_k) \\ &= \big(\frac{1}{3} - \frac{1}{3m}\big) \sum_{j \in J} t_j\ n_j \\ &\leq \big(\frac{1}{3} - \frac{1}{3m}\big)\ C_{\max}(S^*) \end{aligned}$$

Therefore, we have

$$\begin{aligned} C_{\max}(\hat{S}) = C_{\max}(\hat{S}_1) + C_{\max}(\hat{S}_2) &\leq \big(1 + \frac{m-1}{m}\big) C_{\max}(S^*) \ + \ \big(\frac{1}{3} - \frac{1}{3m}\big) C_{\max}(S^*) \\ &= \Big(\big(1 + \frac{m-1}{m}\big) \ + \ \big(\frac{1}{3} - \frac{1}{3m}\big) \Big) C_{\max}(S^*). \end{aligned}$$

□

Now, we present an instance showing that the worst case bound is tight. When the number of machines is two, the theorem states

$$C_{\max}(\hat{S}) \leq \frac{10}{6}\ C_{\max}(S^*).$$

An instance of the tight worst bound for $m = 2$ is illustrated in Figure 4.3. Jobs A and B have $(T - \epsilon)$ processing time with $3N$ units of each required and jobs C, D and F have

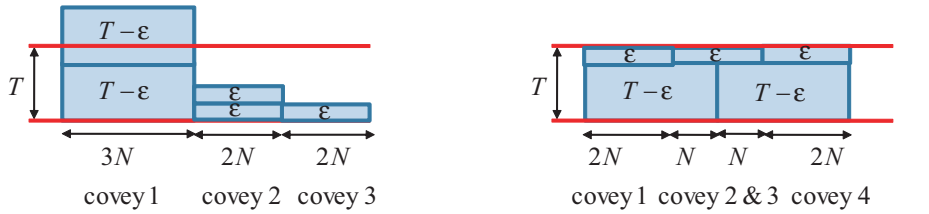
ϵ processing time with $2N$ units required. Schedule S_a of Figure 4.3 (a): the first covey consists of jobs A and B for $3N$ rotations and the second covey consists of jobs C and D for $2N$ rotations, and the third covey consists of only job E for $2N$ rotations. The completion time of schedule S_a is

$$\begin{aligned} C_{\max}(S_a) &= 3N\ \max\{2\ (T-\epsilon),\ T\} + 2N\ \max\{2\ \epsilon,\ T\} + 2N\ \max\{\epsilon,\ T\} \\ &= 6N\ (T-\epsilon) + 2N\ T + 2N\ T \\ &= 10NT - 6N\epsilon. \end{aligned}$$

In the schedule S_b illustrated in Figure 4.3 (b), the first machine produces job A and then B, and the second machine produces job D, E and then F. Then, there are four coveys and the completion time of schedule S_b is

$$\begin{aligned} C_{\max}(S_b) &= 2N\ \max\{(T-\epsilon)+\epsilon,\ T\} + N\ \max\{(T-\epsilon)+\epsilon,\ T\} \\ &\quad +N\ \max\{(T-\epsilon)+\epsilon,\ T\} + 2N\ \max\{(T-\epsilon)+\epsilon,\ T\} \\ &= 6NT. \end{aligned}$$

Therefore, if ϵ goes to zero, $\frac{C_{\max}(S_a)}{C_{\max}(S_b)} = \frac{10}{6}$, showing that the worst case bound is tight. (It also implies that S_b is an optimal schedule and S_a is a worst-possible schedule.)



(a) the worst schedule with $C_{max} = 10NT - 6N\varepsilon$ (b) the optimal schedule with $C_{max} = 6NT$

Figure 4.3: A worst case instance of the *Longest Unit First* when m=2

CHAPTER V

SOLUTION APPROACH OF A REAL-WORLD PROBLEM

In Chapter I, we proposed the following two-phase approach to a real-world float glass problem:

1. **Snap Construction:** Define the *standard snap* for each order by selecting the number and orientation of plates in the order's snaps.

2. **Constructing Cutting and Offloading Schedules:** Construct a schedule for cutting and offloading the correct number of each order's standard snap in a way that satisfies the operational restrictions and machine cycle time limitations.

In this chapter, we propose solution methods for the real world problem, based on the analysis of FGSP and the schedule structure, *coveys*. We provide two methods for the snap construction problem and two methods for constructing cutting and offloading schedules. Computational experiments based on realistic problem instances demonstrate that the proposed methods are very effective. In addition, we provide sensitivity analysis on the number of offloading machines.

5.1 Snap Construction

We do not know how the snap construction decision affects the yield ratio without making the second phase decisions of cutting and offloading schedules. However, we provide two methods, which are expected to help reduce either layout scrap or cycle time scrap in the second phase.

5.1.1 Selection with a Smaller Variance

If all snaps have the same snap width, trivially there is no layout scrap. And, if the variance in snap widths of jobs is bigger, layout scrap is likely to be bigger. Therefore, we prefer a

smaller variance in snap widths in order to reduce layout scrap. Thus, one possibility is to choose the rotation of a plate and the number of plates in a snap so the snap width is closest to the average of the overall maximum and minimum ribbon widths. Then, snap widths are not far away from this middle point, and we expect to have less layout scrap. In the example of Figure 1.5, snap widths of three alternatives are 105, 140, and 120, respectively. Since the snap width of the last alternative is closest to $\frac{100+140}{2} = 120$ where the minimum and maximum ribbon width are 100 and 140 respectively, we would select it to be the order's standard snap.

5.1.2 Selection Considering Machine Balancing Using MIP

In the real world problem, there are two types of offloading machines (HSS and POF) with different requirements for the feasibility of a job being assigned to them. Our second approach chooses standard snaps so that the possible assignment of snaps to the two types of machines is balanced. Figure 5.1 shows results of experiments in which we illustrate the yield ratio according to the change of difference between snaps assigned to HSS machines and snaps assigned to POF machines. In this experiment, three HSS machines and three POF machines are used and 50 instances are tested using our heuristic approaches. Figure 5.1 shows that if the difference between the number of units assigned to HSS machines and that to POF machines is less than 2,000, the yield ratio for most instances is greater than 95%. Meanwhile, if this difference is more than 2,000, the yield ratio is likely to be less than 95%. Therefore, we conclude that the bigger the difference between units assigned to HSS machines and those assigned to POF machines, the lower the yield ratio. Instances for these experiments were generated by drawing jobs from real order data.

In the following mixed integer program, a set of jobs is given, with the size of plates and the required number of plates specified. There is size requirement which specifies the maximum and minimum width of a plate when it is assigned to HSS or POF machines. The formulation needs to choose one alternative among several candidates for snap construction. The objective is to minimize the difference between the number of snaps assigned to HSS

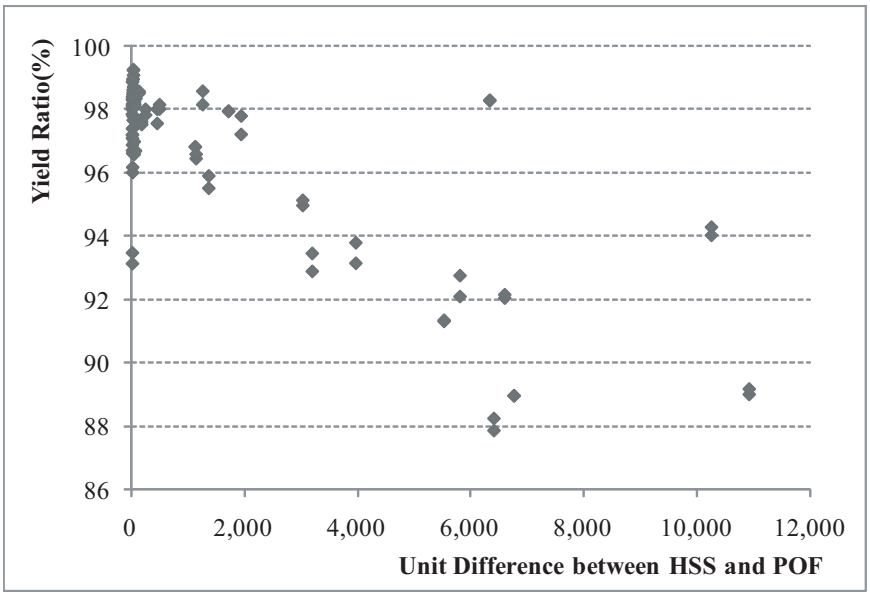


Figure 5.1: Unit(snap) difference between HSS and POF vs. yield ratio

machines and the number of snaps assigned to POF machines. We note that the method of *selection with a smaller variance* does not determine the assignment of jobs to a type of offloading machine.

Parameters

M_p = number of POF machines

M_h = number of POF machines

S_{ji} = Number of snaps for the snap layout alternative i of order j

W_{ji} = Width of a plate for the alternative i of order j

H_{ji} = Height of a plate for the alternative i of order j

P_{ji} = Number of plates for the alternative i of order j

W_{hm} = Minimum width of a plate when it is assigned to HSS

W_{hM} = Maximum width of a plate when it is assigned to HSS

W_{pm} = Minimum width of a plate when it is assigned to POF

W_{pM} = Maximum width of a plate when it is assigned to POF

Decision Variables

$z_{ji} = 1$, if alternative i of order j is selected

$x_{ji} = 1$, if order j and alternative i is assigned to HSS

$y_{ji} = 1$, if order j and alternative i is assigned to POF

$S =$ the difference of number of snaps between HSS and POF

Objective

minimize the difference of number of snaps between HSS and POF

$$\min S \tag{5.1}$$

Constraints

1. select one of the alternatives for each order j

$$\sum_i z_{ji} = 1 \qquad \forall j \in J \tag{5.2}$$

2. select one of HSS and POF machines

$$x_{ji} + y_{ji} = z_{ji} \qquad \forall j \in J \; \forall i \in I \tag{5.3}$$

3. if the POF alternative is selected, the number of plates should be no greater than the number of offloading machines since POF machines pick only one plate

$$P_{ji} \; y_{ji} \leq M \qquad \forall j \in J \; \forall i \in I \tag{5.4}$$

4. size requirement of a plate of HSS machines

$$W_{hm} x_{ji} \leq W_{ji} \qquad \forall j \in J \; \forall i \in I \tag{5.5}$$

$$W_{ji} x_{ji} \leq W_{hM} \qquad \forall j \in J \; \forall i \in I \tag{5.6}$$

5. size requirement of a plate of POF machines

$$W_{pm} y_{ji} \leq W_{ji} \qquad \forall j \in J \; \forall i \in I \tag{5.7}$$

$$W_{ji} y_{ji} \leq W_{pM} \qquad \forall j \in J \; \forall i \in I \tag{5.8}$$

6. S is difference of the number of snaps between HSS and POF machines

$$S \geq \frac{\sum_j \sum_i S_{ji} x_{ji}}{M_h} - \frac{\sum_j \sum_i S_{ji} P_{ji} y_{ji}}{M_p} \tag{5.9}$$

$$S \geq \frac{\sum_j \sum_i S_{ji} P_{ji} y_{ji}}{M_p} - \frac{\sum_j \sum_i S_{ji} x_{ji}}{M_h} \tag{5.10}$$

7. binary and non-negativity restrictions

$$z_{ji} \quad \in \{0,1\} \qquad \forall i \;\; \forall j \tag{5.11}$$

$$x_{ji} \quad \in \{0,1\} \qquad \forall i \;\; \forall j \tag{5.12}$$

$$y_{ji} \quad \in \{0,1\} \qquad \forall i \;\; \forall j \tag{5.13}$$

$$S \quad \geq 0 \tag{5.14}$$

Computational results comparing the two methods of snap construction are provided in Section 5.3.3. In the following sections, we use the method of *selection with a smaller variance* for a snap construction algorithm.

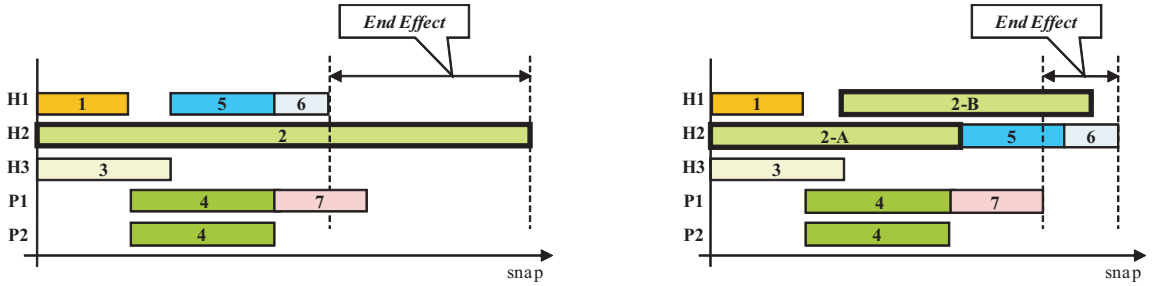
5.2 Constructing Cutting and Offloading Schedules

After snap construction, we determine the sequence of cutting and the assignment of offloading machines. We have a common preprocessing method and two main methods: a MIP approach and a heuristic approach. In the MIP approach, we use a decomposition method and a *reduced snaps procedure* in order to reduce running time. In the heuristic approach, we propose two construction heuristics and two improvement algorithms by local search and dynamic programming.

Preprocessing: *Splitting Long Jobs*

As a preprocessing step, we use the *splitting long jobs* algorithm for helping to balance the workload among machines. At the end of a schedule, we found that because no other jobs remain, a covey often consists of only one or two long jobs. Thus, the sum of the snap times of jobs in the last coveys is less than the cycle time of the offloading machines, which causes the schedule to have cycle time scrap. We refer to this phenomenon as the *end effect*. The

amount of cycle time scrap caused by the end effect often comprises a large portion of a schedule's total scrap.



(a) Job 2 has a large number of snaps.
→ Large end effect.

(b) Job 2 splits into Job 2-A and Job 2-B.
→ Small end effect.

Figure 5.2: Splitting long jobs

In order to reduce the size of large jobs and hence reduce end effect, we split jobs with a large number of snaps into several sub-jobs with fewer snaps. We can only apply this splitting rule as long as the job is large enough to require multiple containers so as not to violate the packaging constraint. For example, if a job of 2,000 snaps requires five containers, we can split it into one job of 1,200 snaps (three containers) and one job of 800 snaps (two containers). Figure 5.2 illustrates how splitting long jobs contributes to decreasing the end effect. In Figure 5.2 (a), Job 2 has a large number of snaps and it is offloaded by an HSS2 machine. As a result, the schedule inevitably has a large end effect. In Figure 5.2 (b), to reduce this end effect, we split Job 2 into Job 2-A and Job 2-B. They are offloaded by HSS1 and HSS2, respectively. The resulting schedule with split jobs has less end effect than the previous schedule.

5.2.1 MIP Approach

We first propose a mixed integer programming formulation based on snap discretization for the float glass problem. Because of the MIP's size and difficulty, a commercial solver such as CPLEX 11 cannot find good solutions within a reasonable running time. To reduce the running time, we use two methods: a *rolling horizon procedure* [15] and a *reduced snaps procedure*. In addition, to reduce the *end effect*, we propose three methods: an *early start of long jobs*, a *long horizon for the last stage*, and *many jobs in the last stage*.

5.2.1.1 Formulation

In machine scheduling problems, a time-indexed formulation is widely used. In the float glass problem, a snap is considered as a time slot. Hence, we propose a snap-indexed formulation based on snap-discretization; a snap is divided into periods and we will consider snap-periods $1, 2, \ldots, S$, where the planning horizon is denoted by S. The objective is to minimize the sum of cycle time scrap and layout scrap and the constraints represents all operational restrictions of the real-world float glass problem.

Data

J = set of jobs

s_j = number of snaps of job j

t_j = time for a snap of job j

w_j = width of job j

l_j = length of a plate of job j

C = cycle time of machines

S = number of time slots (total snaps)

$\mathcal{M}_j^l$ = big M of job j for calculating layout scrap

$\mathcal{M}_j^c$ = big M of job j of slot i for calculating cycle time scrap

M_h = number of HSS machines

M_p = number of POF machines

pl_j = number of plates per one snap of job j

Variables

$x_{ji} = 1$ if job j begins in the ith slot and is offloaded by an HSS machine

$y_{ji} = 1$ if job j is run in the ith slot and is offloaded by an HSS machine

$p_{ji} = 1$ if job j begins in the ith slot and is offloaded by a POF machine

$q_{ji} = 1$ if job j is run in the ith slot and is offloaded by a POF machine

c_{ji} = snap time of job j corresponding to the ribbon width of the ith slot

v_i^c = cycle time scrap in slot i

v_{ji}^l = layout scrap of job j in slot i

u_i = width of ribbon for covey in slot i

Objective

minimize total cycle time scrap and layout scrap

$$\min \sum_{i=1}^{S} v_i^c + \sum_{j \in J} \sum_{i=1}^{S} v_{ji}^l \tag{5.15}$$

Constraints

1. Each job must be scheduled

$$\sum_{i=1}^{S} x_{ji} + \sum_{i=1}^{S} p_{ji} = 1 \qquad \forall j \in J \tag{5.16}$$

2. Determine whether job j is part of the covey in slot i

$$\sum_{k=\max\{1,i-s_j+1\}}^{i} x_{jk} = y_{ji} \qquad \forall i \in 1,\ldots,S \;\; \forall j \in J \tag{5.17}$$

$$\sum_{k=\max\{1,i-s_j+1\}}^{i} p_{jk} = q_{ji} \qquad \forall i \in 1,\ldots,S \;\; \forall j \in J \tag{5.18}$$

3. Covey size equal to or less than the number of machines

$$\sum_{j \in J} y_{ji} \leq M_h \qquad \forall i \in 1,\ldots,S \tag{5.19}$$

$$\sum_{j \in J} pl_j \; q_{ji} \leq M_p \qquad \forall i \in 1,\ldots,S \tag{5.20}$$

4. Each job in the slot i covey must fit on the ribbon

$$w_j(y_{ji} + q_{ji}) \leq u_i \qquad \forall i \in 1,\ldots,S \;\; \forall j \in J \tag{5.21}$$

5. The ribbon width must be non-increasing

$$u_i \geq u_{i+1} \qquad \forall i \in 1,\ldots,S-1 \tag{5.22}$$

6. The ribbon width must be greater than the width of the remaining jobs

$$u_i \geq w_j(1 - \sum_{k=1}^{i} x_{jk} - \sum_{k=1}^{i} p_{jk}) \qquad \forall i \in 1, \ldots, S \;\; \forall j \in J \tag{5.23}$$

7. Snap time of job j corresponding to the ribbon width of slot i

$$c_{ji} \;\geq\; u_i \; t_j/w_j + \mathcal{M}_j^c(y_{ji} + q_{ji} - 1) \qquad \forall i \in 1, \ldots, S \;\; \forall j \in J \tag{5.24}$$

$$c_{ji} \;\leq\; u_i \; t_j/w_j \qquad \forall i \in 1, \ldots, S \;\; \forall j \in J \tag{5.25}$$

$$c_{ji} \;\leq\; \mathcal{M}_j^c(y_{ji} + q_{ji}) \qquad \forall i \in 1, \ldots, S \;\; \forall j \in J \tag{5.26}$$

8. Ccycle time scrap

$$v_i^c \geq C - \sum_{j \in J} c_{ji} \qquad \forall i \in 1, \ldots, S \tag{5.27}$$

9. Layout scrap

$$v_{ji}^l \geq (u_i - w_j) \; t_j/w_j + \mathcal{M}_j^l(y_{ji} + q_{ji} - 1) \qquad \forall i \in 1, \ldots, S \;\; \forall j \in J \tag{5.28}$$

10. Binary and non-negativity restrictions

$$x_{ji} \in \{0, 1\} \qquad \forall i \in 1, \ldots, S \;\; \forall j \in J$$

$$y_{ji} \in \{0, 1\} \qquad \forall i \in 1, \ldots, S \;\; \forall j \in J$$

$$p_{ji} \in \{0, 1\} \qquad \forall i \in 1, \ldots, S \;\; \forall j \in J$$

$$q_{ji} \in \{0, 1\} \qquad \forall i \in 1, \ldots, S \;\; \forall j \in J$$

$$c_{ji} \geq 0 \qquad \forall i \in 1, \ldots, S \;\; \forall j \in J$$

$$v_i^c \geq 0 \qquad \forall i \in 1, \ldots, S$$

$$v_{ji}^l \geq 0 \qquad \forall i \in 1, \ldots, S \;\; \forall j \in J$$

$$u_i \geq 0 \qquad \forall i \in 1, \ldots, S$$

5.2.1.2 Methods for Reducing Running Time

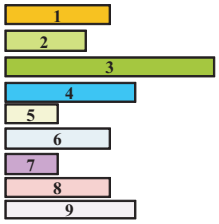
To reduce the running time of the MIP, we propose a *reduced snaps procedure* and a *rolling horizon procedure*. However, we cannot guarantee the optimal solution since those two

methods are heuristic.

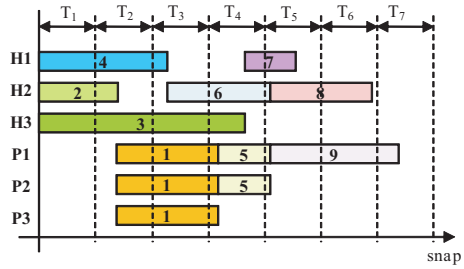
Reduced Snaps Procedure

In the float glass problem, customers usually place a job for a large quantity of plates. After the snap construction step, most jobs need more than 100 snaps. Because of these large quantities, coveys often have many rotations. So, if we scale down the required number of snaps, the main structure of the optimized schedule does not change much. In Figure 5.3 (b), we divide the number of snaps for each job by 3 and then round to get an integer number of snaps, and we find schedules (5.3 (c)) for these jobs with the reduced snaps. In Figure 5.3 (d), the number of snaps is scaled up back to the required number while maintaining the structure of the coveys. The main cutting sequence and assignment of offloading machines is almost the same as the result obtained by solving with the original number of snaps.

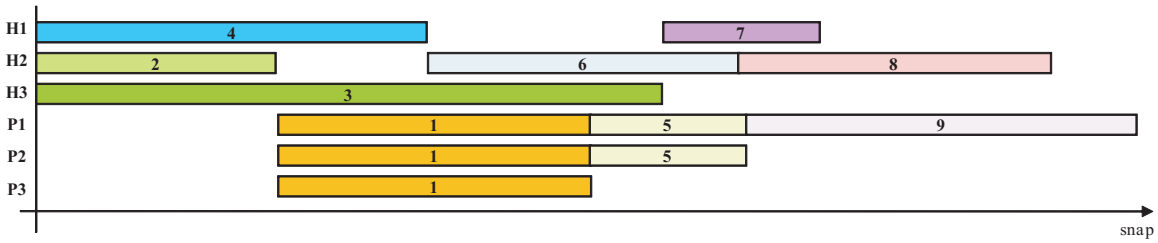
(a) Step 0: Nine jobs are given with original number of snaps



(b) Step 1: Reduce the number of snaps



(c) Step 2: Solve the reduced instance using the rolling horizons



(d) Step 3: Construct the original instance according to the schedule structure obtained in Step 2.

Figure 5.3: *Reduced Snaps Procedure* and *Rolling Horizon Procedure* for the MIP

Rolling Horizon Procedure

A shift schedule usually takes from four hours to three days, and the line produces a snap every 3 or 4 seconds. As a result, one planning horizon of the schedule has a huge number of snap periods, and a mixed integer formulation has huge amount of variables and constraints. The rolling horizon procedure decomposes the problem by periods and determines the schedule up to a given period. Then, the procedure continues to the next period until the end of the periods. In Figure 5.3 (c), the whole problem is divided into seven stages and their schedules are determined from the first period to the last period. Our experiments use 150 snaps per stage.

5.2.1.3 Methods for Reducing the End Effect

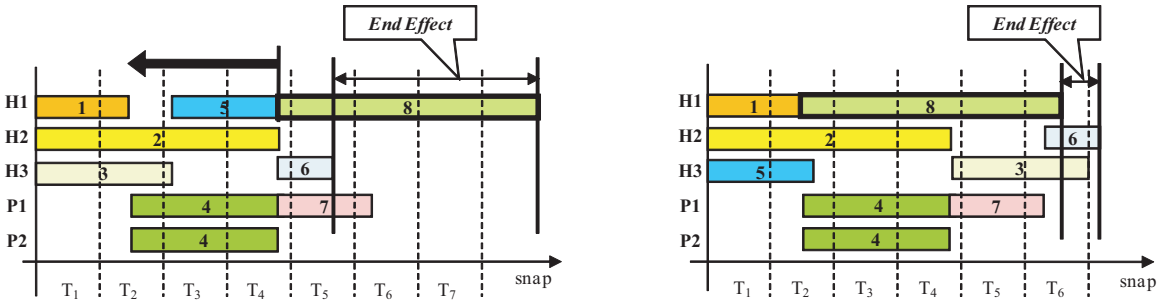
It is hard to avoid the end effect if we use the rolling horizon procedure. However, some phenomena make the end effect larger. We now analyze these bad cases, and propose methods to reduce the end effect

Early Start of Long Jobs

As mentioned earlier, the end effect occurs because few jobs are remaining in the later stages of the rolling horizon procedure. If some of these remaining jobs have a large number of snaps, we have larger end effect. To avoid such a case, we enforce that the jobs with a large number of snaps should be run in an earlier stage. Assume that the period of one stage is p snaps. Let Q be an estimate of stages needed. Assume that one long job has s snaps. Then, this job needs at least $\lceil \frac{s}{p} \rceil$ stages, so it should be started before the $(Q - \lceil \frac{s}{p} \rceil)$th stage so as not to make the end effect larger. In Figure 5.4 (a), Job 8 has a large number of snaps and it starts at stage T_4 causing a large end effect. In Figure 5.4 (b), since this job is started at T_2, the end effect is reduced.

Long Horizon for the Last Stage

So far we have assumed that the size of every stage is the same. If the size is large,

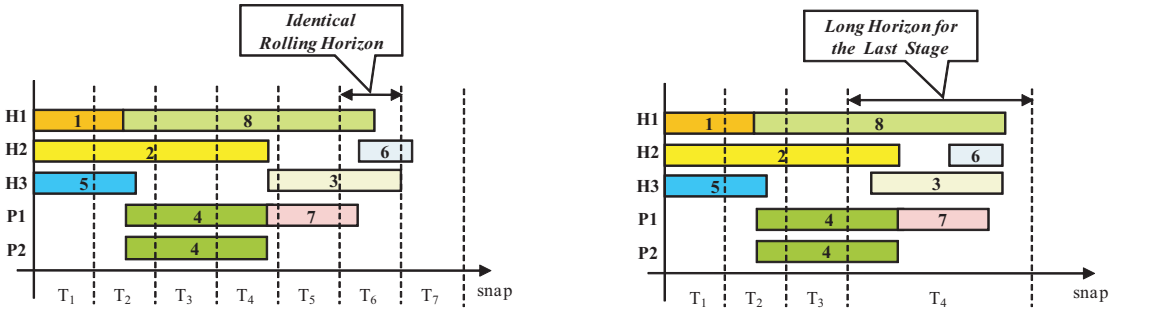


(a) Job 8 with a large number of snaps starts in stage T_4. → Large end effect.

(b) Job 8 starts in the earlier stage of T_2. → Small end effect.

Figure 5.4: Early start of long snaps jobs

experimenting running time can be prohibitive. However, we can find a reasonable stage size satisfying both solution quality and running time. At every rolling horizon, we need to decide which jobs should be run at this time, and the number of jobs are not so many. In the later stages, the number of jobs are fewer leading to a small number of alternative schedules and we are likely to have a large end effect. To fix such a case, we make the last stage larger than the other, which leads to have more alternative schedules, and we can expect to reduce the end effect. In Figure 5.5 (a), stage T_6 only determines the starting point of Job 6 since only Job 6 remains. In Figure 5.5 (b), stage T_4 is the last one and it is bigger than the other stage. In T_4, we need to schedule three jobs: Jobs 3, 6 and 7. Hence, we have more alternatives, and we can reduce the end effect. In our experiments, if the sum of cycle time scrap per stage is greater than $\frac{T}{2}$, we run the remaining jobs in one last long period rolling.



(a) Horizons of all stages are identical. → Large end effect.

(b) The horizon for the last stage is long. → Small end effect.

Figure 5.5: Comparison between identical horizon and long horizon for the last stage

Many Jobs in the Last Stage

The main idea of *many jobs in the last stage* is similar to a *long horizon for the last stage*. However, this method determines the last stage in terms of the number of remaining jobs; if the number of remaining jobs is less than a certain number, we run the last stage for all remaining jobs. We decided a parameter of the level (the number of remaining jobs) as seven jobs from experiments.

5.2.1.4 Computational Results

We tested our algorithms on 150 instances with 50 jobs randomly generated from actual data. For the tests, our algorithms were implemented in C++ with CONCERT technology and executed by CPLEX 11 on a Linux machine with two 2.4 GHz Processors and 2GB RAM. We chose parameters to be close to the environment of a float glass plant: we assumed four HSS and four POF machines are available, the cycle time of both POF and HSS offloading machines is 13 seconds, and the ribbon width varies between 120 and 144 inches.

Table 5.1 shows the quality of the solutions and running time. We show the results of *normal rolling*, *early start of long jobs*, *long horizon for the last stage*, and *many jobs in the last stage*. In all four methods, after running MIP we execute a *push-back* algorithm, which will be discussed in Section 5.2.2.2. We also compare *splitting of long jobs* with no splitting. *Long horizon for the last stage* with *splitting of long jobs* gives schedules with the highest yield ratio.

Table 5.1: Computational results of MIP approach on randomly-generated data

	Yield Ratio(%)		Time(sec)	
Methods	No Splitting	Splitting of Long Jobs	No Splitting	Splitting of Long Jobs
Normal Rolling + Push-Back	93.2	97.4	51	72
Early Start of Long Jobs + Push-Back	95.6	97.4	50	74
Long Horizon for Last Stage + Push-Back	95.3	98.9	69	95
Many Jobs in Last Stage + Push-Back	94.3	97.3	76	103

5.2.2 Heuristic Approaches

Our heuristic approaches consists of two phases: construction and improvement. We provide two construction heuristics, an improvement algorithm by local search, and an improvement algorithm by dynamic programming.

5.2.2.1 Construction

We use greedy construction algorithms to obtain initial schedules. Jobs are prioritized according to some characteristics (discussed below). By priority, they are then assigned to offloading machines in the following way. If the sum of snap times in the current covey is less than the machine cycle time (which would result in this covey having cycle time scrap), then it is myopically beneficial to add another job to the covey. Thus, the next job in the priority list is assigned to that covey if a machine is available to offload it. Otherwise, if the covey already has no cycle time scrap or if no machine is available, this covey is considered "full" and the algorithm advances to the next covey. The process iterates until all jobs are assigned.

Assume that the ribbon width is monotonically non-increasing. Then, wider jobs should be produced earlier; otherwise, narrow jobs will incur layout scrap. Therefore, we first sort the jobs in non-increasing order of width.

It is not uncommon to have many jobs of the same width. We have two different ways of breaking ties between such jobs, leading to two slightly-different construction heuristics. Both tie-breaking methods are designed to help reduce the end effect.

One tie-breaking method is to produce jobs with shorter snap times first, to save longer-snap-time jobs for coveys that might be subject to the end effect; in such a case, the cycle time scrap of the end effect would be smaller. We refer to the heuristic using this tie-breaking method as the Width-Then-Time (WTT) method.

A second tie-breaking method is to begin jobs with more snaps first, so that these jobs are less likely to have long end effects. We refer to the heuristic using this tie-breaking method as the Width-Then-Snaps (WTS) method.

Dynamic Assignment Considering Balancing between HSS and POF

As mentioned earlier, if the assignment of jobs to offloading machines is not balanced between the HSS and POF machines, one machine type may be busy while the other type is idle, thus increasing the chance of incurring cycle time scrap. Some jobs' characteristics require them to be picked by a certain type of machine. However, jobs which satisfy the size requirements of both HSS and POF machines can be offloaded by either type. For these jobs, we decide the assignment of offloading machine type in order to minimize the difference between the total number of snaps offloaded by HSS machines and offloaded by POF machines. In general, snaps with big plates are assigned to POF machines and snaps with small plates are assigned to HSS machines, but snaps with a medium size of plates can be assigned to either HSS or POF machines. Whenever we assign jobs to offloading machines, we record the numbers of snaps assigned to HSS machines and POF machines. When we assign a job with snaps with a medium size of plates, the difference between the number of HSS snaps or the number of POF snaps is greater than a certain threshold, the job is assigned to the machines with a smaller number of snaps.

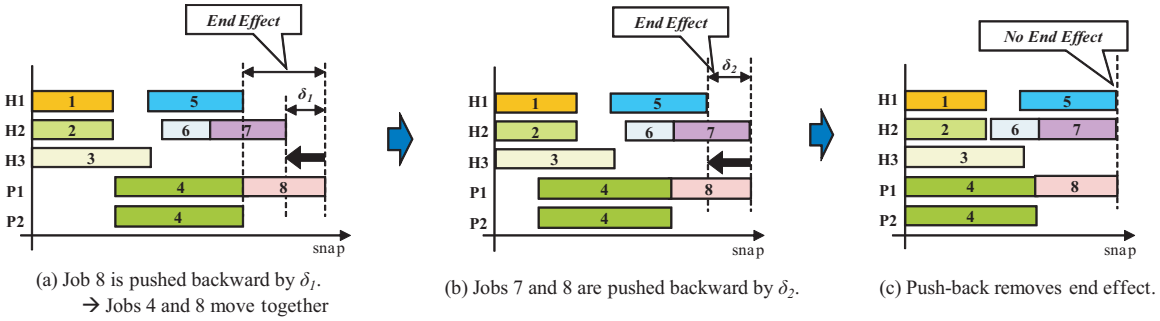
5.2.2.2 Improvement by Local Search

Once an initial schedule has been constructed, we use three local improvement methods, *push-back*, *relocation*, and *exchange*. We first apply push-back to the initial schedule, followed by repeated application of relocation and exchange until there is no more improvement in the yield ratio. The push-back algorithm can be applied to improve any schedule. Therefore, we apply it not only to the initial schedule, but also to all potential relocation and exchange schedules before evaluating their yield ratio.

Push-back of Jobs:

The purpose of the push-back step is to eliminate as much of the end effect as possible. We try to push backward (earlier in time) the jobs located in the later coveys in order to combine them with jobs of the earlier coveys. This can reduce the number of snaps with

few jobs (and consequently shorter total snap times) and decrease the total cycle time scrap at the end of a shift.



(a) Job 8 is pushed backward by δ_1.
→ Jobs 4 and 8 move together

(b) Jobs 7 and 8 are pushed backward by δ_2.

(c) Push-back removes end effect.

Figure 5.6: Iterative push-back algorithm

At each iteration, the algorithm pushes backward the jobs in the last covey by the number of snaps in that last covey. The push-back algorithm is repeatedly applied to the last covey of a schedule as long as the covey currently incurs cycle time scrap and push-back is feasible.

Figure 5.6 shows an example of the push-back algorithm. In Figure 5.6 (a), the last covey of the current schedule consists only of Job 8, and the number of snaps in this covey is δ_1. So, the push-back algorithm moves Job 8 leftward by δ_1 snaps. As a consequence, Job 4 also gets pushed back δ_1 snaps to make room, creating the schedule in Figure 5.6 (b).

Assume now that sum of snap times of Job 7 and Job 8 is less than the machine cycle time. Then, the schedule in Figure 5.6 (b) still has an end effect. Thus, we want to push back Jobs 7 and 8 by δ_2 snaps. Jobs 6 and 4 are also pushed backward to make room, resulting in the schedule shown in Figure 5.6 (c).

Figure 5.6 (c) shows that no end effect now exists, since the sum of snap times of Jobs 5, 7, and 8 is greater than the machine cycle time. However, even if there still was an end effect, no push-back could be done, since there is no room to push Jobs 8 and 4 backward on the first POF machine.

Relocation of Jobs

The relocation algorithm iteratively selects a job and evaluates the benefit of changing its position in the schedule. It tries to relocate the job to the starting point of each covey on

each machine of the same type as the job is currently on. For each possible relocation, we run the push-back algorithm (if necessary) and then compute the yield ratio of the updated schedule. If the new yield ratio is higher than the yield ratio before the relocation, then the new schedule is retained; otherwise, we return to the schedule before the relocation. The algorithm cycles through all jobs, attempting to relocate each one in turn, until a complete cycle is made with no changes.

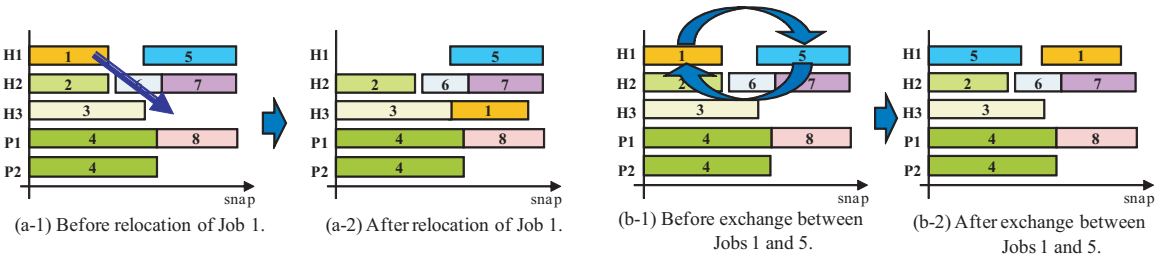


Figure 5.7: Examples of relocation and exchange of jobs

Figures 5.7 (a-1) and (a-2) illustrate one step of the relocation algorithm. Before relocation, Job 1 is offloaded by HSS 1 machine. As the result of relocation, Job 1 is offloaded by HSS 3 right after Job 3 is offloaded.

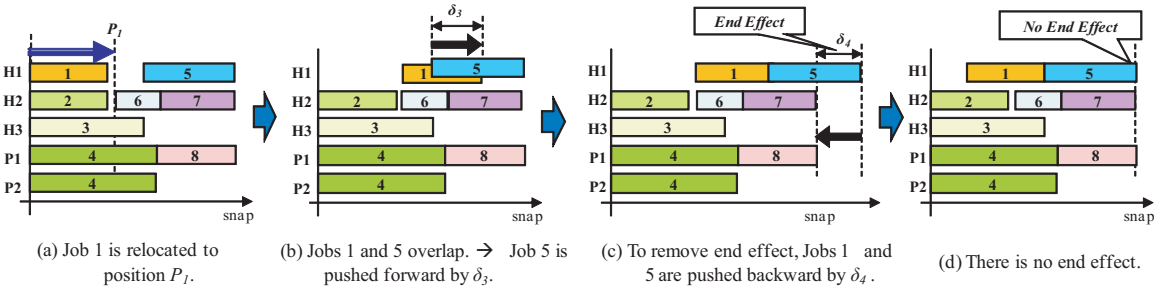


Figure 5.8: Example of push-forward and push-back algorithms in relocation

It might happen that when a job is relocated, it has enough snaps that its end overlaps the scheduled start of the next job on the same machine. In such a case, we first apply a *push-forward* operation. This operation might produce an increased end effect. Therefore, we apply the push-back algorithm after each relocation of jobs in order to remove as much of this end effect as possible. The push-forward and subsequent push-back steps are illustrated in Figure 5.8.

Exchange of Jobs

The exchange algorithm iteratively selects two jobs and evaluates the benefit of exchanging their positions in the schedule. Similar to relocation, jobs are exchanged only between the same type of machine. The push-back algorithm is run after each potential exchange before calculating its yield ratio and comparing with the yield ratio of the current schedule.

Figures 5.7 (b-1) and (b-2) illustrate one step of the exchange algorithm. Before exchange between Jobs 1 and 5, HSS 1 offloads Job 1 first then Job 5. As the result of exchange, HSS 1 offloads Job 5 first then Job 1.

An Illustrative Example

In this section we illustrate our solution approach on an example instance. In this small example, we assume that three HSS machines and three POF machines are available, all with cycle times of 10 seconds. We also assume that the minimum and maximum ribbon widths are 120 and 140 inches. Table 5.2 gives the dimensions and number of plates in each order. For most orders, several snap constructions are possible. The selected standard snap construction for each order (the one whose width is closest to the average of the maximum and minimum ribbon widths in the shift ; in this example, it is closest to 130) is denoted by an asterisk in the "selected snap" column in Table 5.2 and listed in Table 5.3.

First, we assign jobs to machine types. Because the number of plates per snap for Job 1, 5, and 9 is no more than three, we can assign those jobs to POF machines. The other jobs are assigned to HSS machines.

Next, we create a priority list for each construction algorithm. For WTT, the list is $(O4, O2, O3, O1, O6, O5, O7, O8, O9)$, and for WTS the list is $(O3, O4, O1, O2, O6, O5, O9, O8, O7)$. We then apply the construction algorithms to each list of jobs. Figure 5.9 illustrates the schedules after applying the construction algorithms.

The schedules generated by WTT and WTS have end effects that incur significant cycle time scrap. In Figure 5.10, we would like to apply push-back to the schedule generated by WTS, but there is not enough empty space to push Job 8 backward. However, during the relocation phase of local improvement, Job 8 gets relocated to the third HSS machine and

Table 5.2: Alternatives for snap construction (small example)

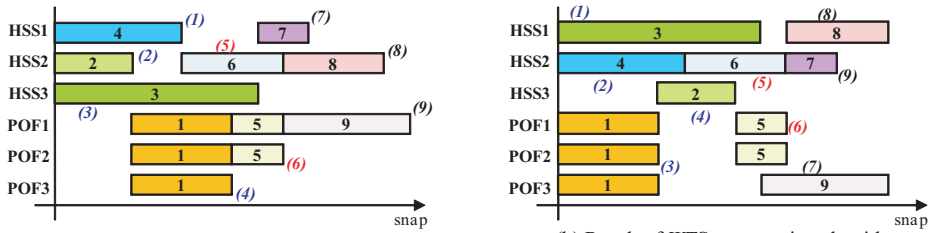
Given customer orders			Alternatives for snap construction			
index	dimension	total plates	width × height	plates	snap width	selected snap
1	44 × 50	600	44 × 50	3	132	*
2	30 × 33	600	30 × 33	4	120	
			33 × 30	4	132	*
3	22 × 45	2,400	22 × 45	6	132	*
			45 × 22	3	135	
4	16.5 × 25	2,000	16.5 × 25	8	132	*
			25 × 16.5	5	125	
5	60 × 65	200	60 × 65	2	120	
			65 × 60	2	130	*
6	32.5 × 35	800	32.5 × 35	4	130	*
			35 × 32.5	4	140	
7	16 × 20	800	16 × 20	8	128	*
			16 × 20	9	144	
			20 × 16	6	120	
			20 × 16	7	140	
8	30 × 32	800	30 × 32	4	120	
			32 × 30	4	128	*
9	55 × 128	250	128 × 55	1	128	*

Table 5.3: Result of snap construction (small example)

index	total plates	snap width	width × height	plates	# of snaps	snap time
1	600	132	44 × 50	3	200	5.0
2	600	132	33 × 30	4	150	3.0
3	2,400	132	22 × 45	6	400	4.5
4	2,000	132	16.5 × 25	8	250	2.5
5	200	130	65 × 60	2	100	6.0
6	800	130	32.5 × 35	4	200	3.5
7	800	128	16 × 20	8	100	2.0
8	800	128	32 × 30	4	200	3.0
9	250	128	128 × 55	1	250	5.5

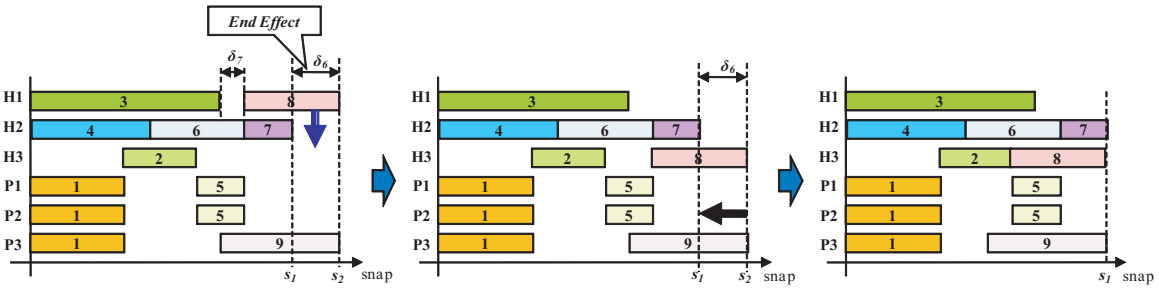
then gets pushed back with Job 9.

Figure 5.11 shows a similar situation with the schedule generated by *WTT*. In Figure 5.11, Job 8 cannot be pushed backward because there is not enough space, i.e. $\delta_2 > \delta_3$. However, when Job 8 and Job 7 are exchanged, iterative push-back can be successfully applied to remove the end effect.



(a) Result of *WTT* construction algorithm. (b) Result of *WTS* construction algorithm.

Figure 5.9: Result of greedy construction algorithms (small example)

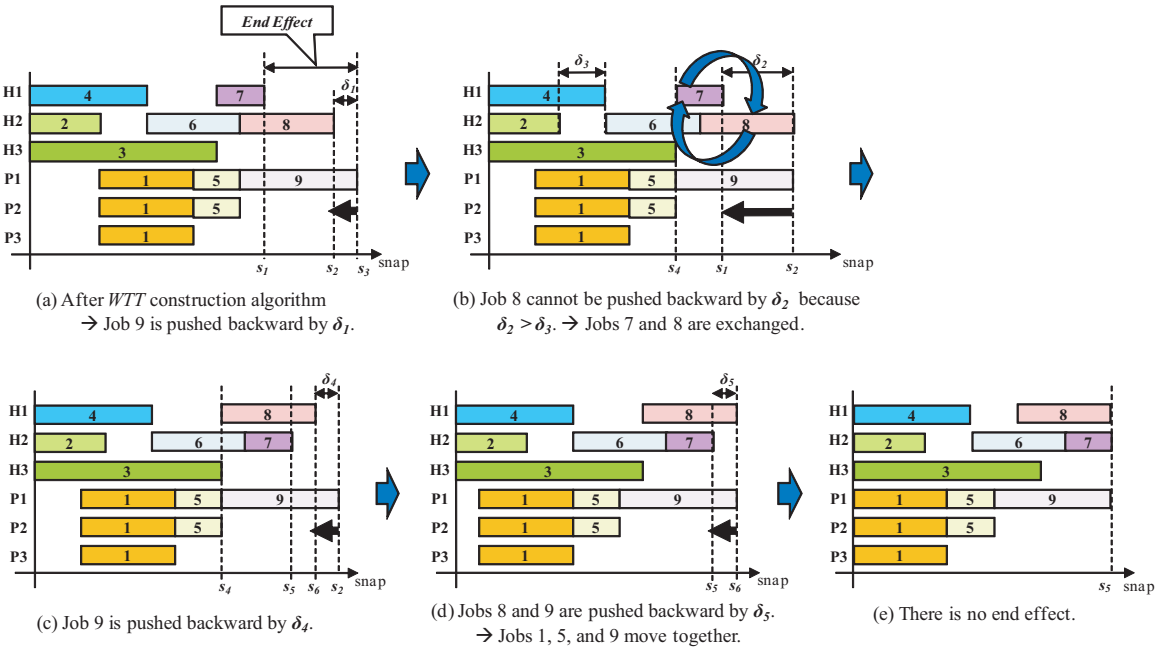


(a) After *WTS* construction algorithm
→ Job 8 cannot be pushed backward.
→ Job 8 is relocated.

(b) Jobs 8 and 9 are pushed backward by δ_6.

(c) There is no end effect.

Figure 5.10: Local improvement after *WTS* algorithm



(a) After *WTT* construction algorithm
→ Job 9 is pushed backward by δ_1.

(b) Job 8 cannot be pushed backward by δ_2 because $\delta_2 > \delta_3$. → Jobs 7 and 8 are exchanged.

(c) Job 9 is pushed backward by δ_4.

(d) Jobs 8 and 9 are pushed backward by δ_5.
→ Jobs 1, 5, and 9 move together.

(e) There is no end effect.

Figure 5.11: Local improvement after *WTT* algorithm

5.2.2.3 Improvement by Dynamic Programming

The local search methods such as relocation and exchange of jobs try to change the sequence of jobs, assignment of offloading machines to jobs, and the starting points of jobs. As a result, the structure of coveys also change. In this section, when the sequence of jobs and the assignment of offloading machines to jobs are fixed (or already determined), we want to find the optimal starting point of each job using a dynamic programming approach. We decide starting points of the jobs assigned to only one machine, say M_d. The starting points of jobs assigned to the other machines are fixed. In Figure 5.12, we would like to find the optimal position of jobs assigned to the HSS4 machine. Positions of the jobs assigned to the HSS1, HSS2, and HSS3 machines are fixed.

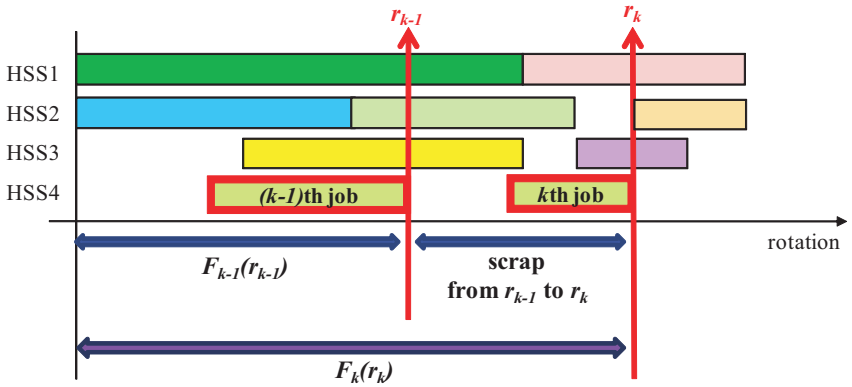


Figure 5.12: Dynamic programming for deciding optimal positions of jobs

The stage of this dynamic program is the number of jobs assigned to offloading machine M_d, denoted by k. The state space is r_k, where r_k is the ending point of the kth job on machine M_d. Since we know the units of each job, once the ending point of a job is determined, we know the starting point of the job. Let $F_k(r_k)$ be the least possible scrap when the kth job ends at the r_kth rotation on machine M_d. We have the following recursion:

$$F_k(r_k) = \min_{r_{k-1}} \Big\{ F_{k-1}(r_{k-1}) + (\text{scrap from } r_{k-1} \text{ to } r_k) \Big\}$$

In words, this recursion means the following: in order to find the optimal position of the kth job, we consider the scrap of the previous stage with the $(k-1)$st job. The first part is the value of the previous stage where the $(k-1)$st job ends at the r_{k-1}st rotation. The

second part is the amount of scrap between the r_{k-1}st rotation and the r_kth rotation when the kth job ends at the r_kth rotation. We find r_k so that the sum of the first part and second part is minimized.

Now, consider the complexity of the dynamic program. Assume that n jobs are assigned to machine M_d and at most R rotations exist. When choosing the kth job, we consider $F_{k-1}(r_{k-1})$ and the alternatives for r_{k-1} are at most r rotations. For each r_{k-1}, we consider all possible position of r_k. Therefore, for each stage, it takes $O(r^2)$ time. Since there are n stages, the total running time is $O(nr^2)$.

When we apply this dynamic program to jobs on POF machines, we consider all POF machines together because some jobs can be assigned to multiple POF machines. The same recursion can be applied, but the feasible range where r_k can be located is smaller than that of the dynamic program for one POF machine. We apply this dynamic program for one machine iteratively to all the machines. We run this dynamic program for HSS1, and then for HSS2, and for HSS3, and for HSS4, and then for POFs 1,2,3 and 4 together. Then, if there is any improvement in the yield ratio, we continue to apply it for HSS1, HSS2, and so on. If there is no improvement in the yield ratio, we stop the procedure.

5.2.2.4 Computational Results

We tested our algorithms on actual shift data from the float glass plant we studied, and we compared the yield ratio from our solutions to the yield ratio from theirs. We also tested our algorithms on 250 additional instances randomly generated from actual data.

Table 5.4 shows the quality of the solutions for three actual instances of company order data. The three data sets have 36, 40, and 49 jobs. After splitting long jobs (as described earlier), the number of jobs increases to 55, 46, and 57. In all three instances, the heuristics achieve between 99.2 and 99.6 percent yield in less than 80 seconds, while the yield ratio of the actual shift schedule previously run by the company was 94.8 - 97.4 percent.

For further validation, we conducted an experiment on randomly generated instances, with each job taken from the pool of actual order data to make them realistic. We generated

Table 5.4: Computational results on actual data

	Actual schedule	Balancing+WTT+local search		Balancing+WTS+local search	
Instance	Yield ratio(%)	Yield ratio (%)	Time (sec)	Yield ratio(%)	Time (sec)
Data Set 1	95.3	99.6	70	99.2	79
Data Set 2	97.4	99.2	26	99.4	23
Data Set 3	94.8	99.3	63	99.5	51

250 instances, 50 each with 40, 50, 60, 70, and 80 jobs. Tables 5.5 and 5.6 report heuristic yield ratios broken down by the pieces of the overall solution approach that we used.

Table 5.5: Yield ratio(%) for randomly-generated instances, without splitting of long jobs

	No splitting of long jobs							
	WTT construction				WTS construction			
	No balancing		Balancing		No balancing		Balancing	
# of jobs	WTT	+ local	WTT	+ local	WTS	+ local	WTS	+ local
40	93.3	97.7	93.6	98.0	93.6	97.9	93.5	98.0
50	95.1	98.2	95.2	98.2	96.1	98.3	96.4	98.5
60	95.1	98.6	95.5	98.6	97.0	98.6	96.7	98.6
70	96.2	98.8	96.5	99.0	96.9	98.8	97.3	99.0
80	96.9	98.7	97.2	99.0	97.3	98.9	97.8	99.2
average	95.3	98.4	95.6	98.5	96.2	98.5	96.3	98.6
std. dev.	4.7	1.4	4.6	1.3	4.4	1.3	4.5	1.3
max	99.8	99.8	99.7	99.8	99.8	99.8	99.8	99.9
min	67.5	90.5	72.7	90.5	67.6	90.4	70.6	90.4

Table 5.6: Yield ratio(%) for randomly-generated instances, with splitting of long jobs

	With splitting of long jobs							
	WTT construction				WTS construction			
	No balancing		Balancing		No balancing		Balancing	
# of jobs	WTT	+ local	WTT	+ local	WTS	+ local	WTS	+ local
40	97.3	98.8	97.7	99.0	97.6	98.8	98.1	99.0
50	98.2	99.1	98.4	99.2	98.3	99.0	98.6	99.2
60	98.5	99.2	98.6	99.2	98.5	99.1	98.7	99.2
70	98.2	99.1	98.7	99.3	98.2	99.1	98.7	99.3
80	98.1	99.1	98.5	99.3	98.2	99.1	98.7	99.3
average	98.1	99.1	98.4	99.2	98.1	99.0	98.5	99.2
std. dev.	2.1	0.7	1.5	0.4	2.0	0.8	1.3	0.4
max	99.8	99.8	99.8	99.8	99.9	99.9	99.9	99.9
min	86.9	94.3	88.2	97.8	88.1	94.3	90.8	97.5

As the results shown in Tables 5.5 and 5.6 demonstrate, splitting long jobs and using improvement heuristics both have significant positive impact on the overall solution quality. Although balancing has only a small effect on average, its improvement seems to be concentrated on instances where the overall solution quality without balancing is worst. For example, in the instance where our heuristics perform worst, balancing improves the yield ratio by over 3 percent. Thus, it is also an effective part of our overall solution approach.

Although the two construction algorithms WTT and WTS give very similar average solution quality, their results on individual instances can sometimes vary in yield ratio. Therefore, since the algorithms run quickly relative to the required solution time, we recommended to the manufacturer that when possible, they use both algorithms and select whichever of the two schedules has a higher yield ratio. Table 5.7 shows the results of using this strategy, which improves performance by 0.1 percent.

Table 5.7: Yield ratio (better of WTT and WTS) and running time

	Splitting + balancing + construction + local search			
	WTT	WTS	Better of two solutions	
# of jobs	Yield ratio (%)	Yield ratio	Yield ratio (%)	Total running time (sec)
40	99.0	99.0	99.1	37
50	99.2	99.2	99.3	83
60	99.2	99.2	99.3	182
70	99.3	99.3	99.4	355
80	99.3	99.3	99.4	614
average	99.2	99.2	99.3	254
std. dev.	0.4	0.4	0.4	301
max	99.8	99.9	99.9	2,131
min	97.8	97.5	97.9	12

In Table 5.8, since our iterative dynamic program can reduce the end effect, we compare it with the result of the push-back algorithm. We tested on 50 instances with 40 jobs. After applying the construction phase algorithms, the assignment of offloading machines and the sequence of jobs on one machine are fixed. Then, we apply the iterative dynamic program. For the WTS construction algorithm, the yield ratio of the dynamic program is 0.4% greater than that of the push-back algorithm, but for the WTT construction algorithm, the yield ratio of the dynamic program is 0.2% less than that of the push-back algorithm.

Their difference is small, but the dynamic program takes more running time. Hence, as an alternative to the push-back algorithm for reducing the end-effect, the dynamic program does not have much advantage.

Table 5.8: Results of dynamic programming I: comparison with push-back algorithm

	Yield ratio(%)		Running time(sec)	
	Construction + Push-Back	Construction +DP	Construction + Push-Back	Construction +DP
WTS	97.7	98.1	0.005	23
WTT	98.1	97.9	0.004	24

In Table 5.9, we report results of improvement by dynamic programming when it is applied after all heuristic approaches including local search. For the WTS construction algorithm, the yield ratio improves by 0.014%, and for the WTT construction algorithm, the yield ratio improves by 0.005%. These are not big improvements, but the iterative dynamic program helps to find a slightly better solution.

Table 5.9: Results of dynamic programming II: applied after local search

	Yield ratio(%)		Running time(sec)	
	Construction +LocalSearch	Construction +LocalSearch+DP	Construction +LocalSearch	Construction +LocalSearch+DP
WTS	99.025	99.039	32	56
WTT	99.016	99.021	30	54

5.3 Sensitivity Analysis

In the computational experiments, we use the exact number of HSS and POF machines as are used in the float glass plant we studied; there are four HSS machines and four POF machines. However, our algorithm might work differently if the number of offloading machines changes in the cases of machine breakdown or machine maintenance. Moreover, when initially building a float glass plant, a manufacturer must decide how many offloading machines to purchase. For upgrading production lines too, it is important to know the optimal number of offloading machines in order to maximize profit. To answer such questions, we

conduct sensitivity analysis on the number of offloading machines.

5.3.1 Sensitivity Analysis on the Number of Offloading Machines

Table 5.10 shows the result of sensitivity analysis on the number of offloading machines. We tested on the 50 instances with 60 jobs, described in Section 5.2.2.4. We vary the number of HSS machines and POF machines from two to four, and for all possible combinations, we apply our heuristic approaches explained earlier. When the number of POF machines is fixed as two, the yield ratios are 88.8%, 95.9%, and 97.7% when the numbers of HSS machines are 2, 3, and 4, respectively using the snap construction with a smaller variance. Furthermore, when the number of HSS machines is fixed as two, the yield ratios are 88.8%, 89.8%, and 93.3% when the numbers of POF machines are 2, 3, and 4, respectively using the snap construction with a smaller variance. Since an HSS machine is more efficient than a POF machine, we observe that the number of HSS machines is more critical in maximizing the yield ratio.

Table 5.10: Sensitivity analysis

		Construction of a smaller variance			MIP construction for Balancing		
# of HSS	# of POF	Layout scrap(%)	Cycle time scrap(%)	Yield ratio(%)	Layout scrap(%)	Cycle time scrap(%)	Yield ratio(%)
2	2	2.7	8.5	88.8	2.5	6.2	91.2
2	3	2.6	7.6	89.8	2.9	7.5	89.6
2	4	2.1	4.6	93.3	2.2	5.1	92.7
3	2	2.3	1.9	95.9	2.3	1.7	96.0
3	3	2.3	2.3	95.4	2.2	1.3	96.6
3	4	1.4	0.3	98.3	1.5	0.3	98.2
4	2	1.6	0.7	97.7	2.1	1.0	97.0
4	3	1.8	1.4	96.8	1.9	0.6	97.4
4	4	0.7	0.03	99.2	1.1	0.01	98.9
	average	1.9	3.0	95.0	2.1	2.6	95.3

5.3.2 Analysis on Ratios of Layout Scrap and Cycle Time Scrap

In float glass manufacturing, cycle time scrap is more critical than layout scrap because we have the flexibility to change ribbon width. Purchasing more machines helps to reduce the amount of cycle time scrap. In Table 5.10, when we have fewer offloading machines, the ratio of cycle time scrap is bigger than the ratio of layout scrap. If we purchase more offloading machines, the ratio of cycle time scrap decreases and the overall yield ratio increases. In the current manufacturing environment with four HSS and four POF machines, we see that the ratio of cycle time scrap is less than 0.03%. Most of the scrap comes from layout scrap. Hence, the plant does not have a big incentive to purchase more offloading machines.

5.3.3 Comparison between Snap Construction Algorithms

We report the results of two snap construction methods: selection with a smaller variance, and snap construction MIP considering balancing between HSS and POF. The running time of the snap construction MIP is less than 2 seconds so time is not much of a factor. In Table 5.10, when the number of HSS and POF machines are both four, the method of smaller variance is 0.3% better than that of MIP for balancing. We have more layout scrap than cycle time scrap when we have enough offloading machines. The method of smaller variance helps to reduce layout scrap, thus its yield ratio is better in the case of four HSS and four POF machines. The average yield ratio of a smaller variance is 95.0 %, and that of MIP for balancing is 95.3%. When the plant does not enough offloading machines, cycle time is more critical. Thus, for such an environment, the method of MIP for balancing is more effective because the balancing issue has big effect on the cycle time scrap.

CHAPTER VI

CONCLUSIONS AND FUTURE RESEARCH

Motivated by operational issues in real-world glass manufacturing, this thesis addressed a problem of laying out and sequencing the orders so as to minimize wasted glass, called *scrap*. There are two different types of scrap, *layout scrap* and *cycle time scrap*, and a number of operational restrictions are imposed by the manufacturer because of the limitations of the glass-making process and the equipment involved. These manufacturing characteristics relate this problem to existing works such as two-dimensional cutting problems, the hybrid flow shop, the no-wait flow shop, and cyclic scheduling problems. Each of these problems addresses an individual aspect of our problem, but no literature addresses the full complexity of our problem. We proposed a two-phase approach: snap construction and constructing cutting and offload schedules. Regarding the second phase problem, we introduced FGSP (float glass scheduling problem), and provided its solution structure, called *coveys*. We analyzed simple sub-models of FGSP considering the main elements: time, unit, and width. We discussed the complexity for the simple models as well as the full model. Since FGSP is NP-complete, we proposed a heuristic algorithm, *Longest Unit First (LUF)*, and analyzed the worst case performance of the algorithm in terms of the quality of solutions; the worst case performance bound is $\left\{\left(1+\frac{m-1}{m}\right)+\left(\frac{1}{3}-\frac{1}{3m}\right)\right\}$.

For the overall float glass problem in a realistic setting, we developed a number of approaches. The heuristic approaches performed better than a MIP approach. The *end effect* is a main reason for low yield ratios in actual manufacturing plants. Our iterative *push-back* algorithm successfully reduced or almost eliminated this *end effect* and contributed to increasing the yield ratio. Several local search procedures that maintain feasibility and increase yield were developed for FGSP. The MIP approach is based on snap discretization and has too many variables to solve in a reasonable amount of time. Though we used several heuristics to solve the MIP, they were less effective at improving the yield ratio quickly.

Sensitivity analysis helps to give managerial insights about questions such as what the optimal number of offloading machines is. The plant that we studied can achieve a yield ratio over 99% (based on our experiments) with the current number of offloading machines. At this 99% level of yield, the ratio of cycle time scrap is only 0.03%. Therefore, we can conclude that the plant does not need to purchase more offloading machines to reduce cycle time scrap. We also observed that the ratio of layout scrap is bigger when the plant has enough machines, but the ratio of cycle time scrap is bigger when the plant has fewer machines. Therefore, from the results of two snap construction methods, when the plant has enough offloading machines, we recommend the method of *selection with a smaller variance* to reduce layout scrap. Meanwhile, when the plant has fewer offloading machines in operation, we recommend the method of *MIP considering machine balancing* to reduce cycle time scrap.

Our proposed heuristic algorithm, *LUF*, considers only the unit element because it is motivated by the parallel machine scheduling problem. To extend this, an algorithm that can consider the time and the unit elements together might lead to a better worst case bound than the *LUF* algorithm. Another interesting future research is a topic to integrate the snap construction phase and the scheduling phase. We have observed that snap construction affects the overall yield ratio. However, it was hard to analyze how it exactly relates to the overall objective since our approach is separated into two phases. Because our real-world solutions were so close to optimality using a two-phase approach, we did not investigate deciding snap construction and scheduling simultaneously. However, an integrated approach might improve the overall yield for instances where the two-phase approach is less successful.

APPENDIX A

TRIVIAL SOLUTION FOR THE TIME AND UNIT MODEL

The *time and unit* model has a trivial optimal solution in the following case when the number of machines is two.

Lemma 16. *When the number of machines is two, if the number of units of one job is no less than the sum of the number of units of all the other jobs, the optimal schedule of the* time and unit *model is that the job with the longest unit is assigned to one machine and the other jobs are assigned to another machine. That is, in the optimal schedule, each covey should have the job with the longest unit.*

Proof. For a contradiction, assume that in the optimal schedule there exists a covey that

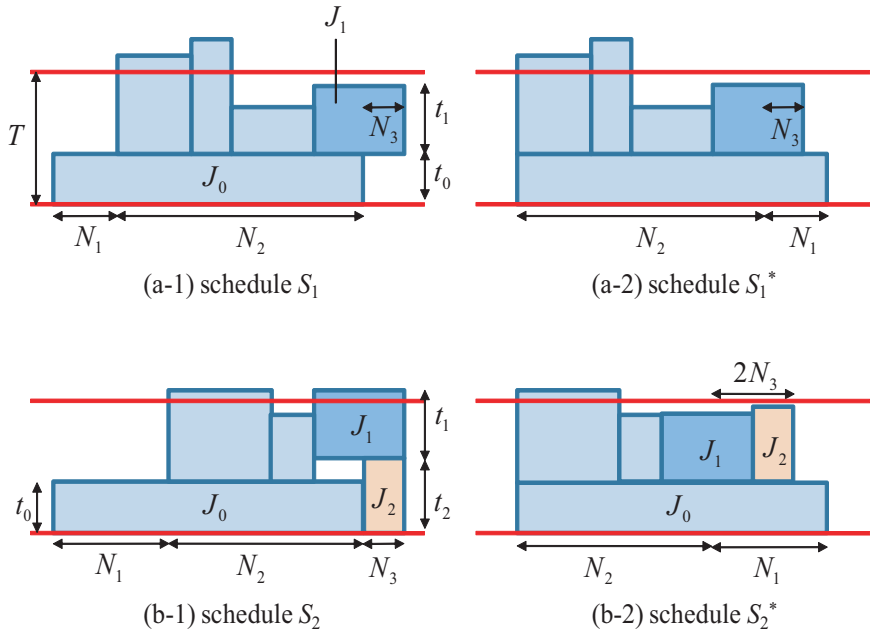


Figure A.1: Trivial optimal schedule of the *time and unit* model when the number of machines is two.

does not have the job with the longest unit. Suppose that J_0 is the job with the longest unit and the processing time of J_0 is t_0. There are two cases: (i) there exists a covey which

has only one job other than J_0 and (ii) there exists a covey which has two jobs other than J_0. For the first case, one covey of the optimal schedule S_1 has only one job J_1 with time t_1 illustrated in Figure A.1 (a-1). However, in Figure A.1 (a-2), we have another schedule S_1^* which has the less completion time than the schedule S_1 because of the following inequalities:

$$\begin{aligned}
C_{\max}(S_1) &= N_1 \max(t_0, T) + N_3 \max(t_1, T) + C_{\max}(N_2 \text{ area}) \\
&= (N_1 - N_3) \max(t_0, T) + N_3 \max(t_0, T) + N_3 \max(t_1, T) + C_{\max}(N_2 \text{ area}) \\
&\geq (N_1 - N_3) \max(t_0, T) + N_3 \max(t_0 + t_1,\ T) + C_{\max}(N_2 \text{ area}) \\
&\qquad (\because \text{Lemma 17 in Appendix B}) \\
&= C_{\max}(S_1^*)
\end{aligned}$$

For the second case, one covey of the optimal schedule S_2 has two jobs J_1 and J_2 illustrated in Figure A.1 (b-1). However, in Figure A.1 (b-2), we have another schedule S_2^* which has the less completion time than the schedule S_2 because of the following inequalities:

$$\begin{aligned}
C_{\max}(S_2) &= N_1 \max(t_0, T) + N_3 \max(t_1 + t_2,\ T) + C_{\max}(N_2 \text{ area}) \\
&= (N_1 - 2N_3) \max(t_0, T) + 2N_3 \max(t_0, T) + N_3 \max(t_1 + t_2,\ T) + C_{\max}(N_2 \text{ area}) \\
&\geq (N_1 - 2N_3) \max(t_0, T) + N_3 \max(t_0 + t_1,\ T) + N_3 \max(t_0 + t_2,\ T) + C_{\max}(N_2 \text{ area}) \\
&\qquad (\because \text{Lemma 18 in Appendix B}) \\
&= C_{\max}(S_2^*)
\end{aligned}$$

Therefore, in the optimal schedule, all coveys should have the job with the longest unit J_0. □

APPENDIX B

PROPERTIES RELATED TO A MAX FUNCTION

The following lemmas related to the max function is used in the proof of **Lemma** 16 for the case of the trivial optimal solution.

Lemma 17.

$$\max(A,T) + \max(B,T) \quad \geq \quad \max(A+B,\ T)$$

where $A, B, T \geq 0$.

Proof.

$$\begin{aligned}
\max(A+B,\ T) &= \max(A+B-B,\ T-B)+B \qquad (\because B \geq 0) \\
&= \max(A,\ T-B)+B \\
&\leq \max(A,\ T)+B \\
&\leq \max(A,\ T)+\max(B,\ T)
\end{aligned}$$

□

Lemma 18.

$$\max(t_1+t_0,\ T)+\max(t_2+t_0,\ T) \leq \max(t_1+t_2,\ T)+2\max(t_0,T)$$

where $t_0, t_1, t_2, T \geq 0$.

Proof. Without loss of generality, assume that $t_1 \geq t_2$.

Case 1: $t_0 \geq t_1 \geq t_2$

$$
\begin{aligned}
& \max(t_1 + t_0,\ T) + \max(t_2 + t_0,\ T) \\
= \ & \max(t_1 + t_0 - t_1,\ T - t_1) + t_1 + \max\big(t_2 + t_0 - (t_0 - t_1),\ T - (t_0 - t_1)\big) + (t_0 - t_1) \\
& \qquad \big(\because t_1 \geq 0,\ (t_0 - t_1) \geq 0\ \big) \\
\leq \ & \max(t_0,\ T) + t_0 + \max(t_1 + t_2,\ T) \\
& \qquad \big(\because \max(t_0,\ T - t_1) \leq \max(t_0,\ T),\quad \max\big(t_1 + t_2,\ T - (t_0 - t_1)\big) \leq \max(t_1 + t_2,\ T)\ \big) \\
\leq \ & 2\max(t_0,\ T) + \max(t_1 + t_2,\ T) \qquad \big(\because t_0 \leq \max(t_0,\ T)\ \big)
\end{aligned}
$$

Case 2: $t_1 \geq t_0 \geq t_2$

$$
\begin{aligned}
& \max(t_1 + t_0,\ T) + \max(t_2 + t_0,\ T) \\
= \ & \max\big(t_1 + t_0 - (t_0 - t_2),\ T - (t_0 - t_2)\big) + (t_0 - t_2) + \max(t_2 + t_0 - t_2,\ T - t_2) + t_2 \\
& \qquad \big(\because (t_0 - t_2) \geq 0,\ t_2 \geq 0\ big) \\
\leq \ & \max(t_1 + t_2,\ T) + t_0 + \max(t_0,\ T) \\
& \qquad \big(\because \max\big(t_1 + t_2,\ T - (t_0 - t_2)\big) \leq \max(t_1 + t_2,\ T),\quad \max(t_0,\ T - t_2) \leq \max(t_0,\ T)\ \big) \\
\leq \ & \max(t_1 + t_2,\ T) + 2\max(t_0,\ T) \qquad \big(\because t_0 \leq \max(t_0,\ T)\ \big)
\end{aligned}
$$

Case 3: $t_1 \geq t_2 \geq t_0$

$$
\begin{aligned}
& \max(t_1 + t_0,\ T) + \max(t_2 + t_0,\ T) \\
= \ & \max\big(t_1 + t_0 - t_1,\ T - t_1\big) + t_1 + \max(t_2 + t_0 - t_2,\ T - t_2) + t_2 \qquad (\because t_1 \geq 0,\ t_2 \geq 0\ ) \\
\leq \ & \max(t_0,\ T) + \max(t_0,\ T) + t_1 + t_2 \\
& \qquad \big(\because \max(t_0,\ T - t_1) \leq \max(t_0,\ T),\quad \max(t_0,\ T - t_2) \leq \max(t_0,\ T)\ \big) \\
\leq \ & 2\max(t_0,\ T) + \max(t_1 + t_2,\ T) \qquad \big(\because t_1 + t_2 \leq \max(t_1 + t_2,\ T)\ \big)
\end{aligned}
$$

□

REFERENCES

[1] Beasley, J. E. (1985) An Exact Two-Dimensional Non-Guillotine Cutting Tree Search Procedure. *Operations Research*, **33**, 49–64.

[2] Birewar, D.B., and Grossmann, I.E. (1989) Incorporating Scheduling in the Optimal Design of Multiproduct Batch Plants *Computers and Chemical Engineering*, **13**, 141–161.

[3] Birewar, D.B., and Grossmann, I.E. (1989) Efficient Optimization Algorithms for Zero-Wait Scheduling of Multiproduct Batch Plants *Industrial and Engineering Chemistry Research*, **28**, 1333–1345.

[4] Dash, S., Kalagnanam, J., Reddy, C., and Song, S. H. (2007) Production Design for Plate Products in the Steel Industry. *IBM Journal of Research and Development*, **51**, 345–362.

[5] Garey, M.R. and Johnson, D.S. (1979) *Computers and Intractability: A guide to the theory of NP-completeness*. W.H. Freeman and Co.

[6] Gilmore, P. and Gomory, R. (1961) A Linear Programming Approach to the Cutting Stock Problem. *Operations Research*, **9**, 849–859.

[7] Gilmore, P. and Gomory, R. (1963) A Linear Programming Approach to the Cutting Stock Problem–Part II. *Operations Resesearch*, **11**, 863–888.

[8] Glass on Web. "World Demand for Flat Glass." Available from http://www.glassonweb .com/articles/article/301/. Retrieved 28 November 2010.

[9] Graham, R.L. (1969) Bounds on Multiprocessing Timing Anomalies. *SIAM Journal of Applied Mathematics*, **17**, 263–269.

[10] Gupta, J. N. D. and Tunc, E. A. (1991) Schedules for a Two Stage Hybrid Flowshop with Parallel Machines at the Second Stage. *Int. J. Prod. Res.*, **29**, 1489–1502.

[11] Grenzebach Maschinenbau GmbH. "Glass Technology: Robot." Available from http://www.grenzebach.com/index.php/eng/produkthauptordner/glas_technologie/roboter. Retrieved 28 November 2010.

[12] Hall, N.G., and Sriskandarajah, C. (1996) A Survey of Machine Scheduling Problems with Blocking and No-wait in Process *Operations Research*, **44**, 510–525.

[13] Linn, R. and Zhang, W. (1999) Hybrid Flow Shop Scheduling: A Survey. *Computers and Industrial Engineering*, **37**, 57–61.

[14] McCormick, S.T., Pinedo, M.L., Shenker, S., and Wolf, B. (1989) Sequencing In an Assembly Line with Blocking to Minimize Cycle Time. *Operations Research*, **37**, 925–935.

[15] Pinedo, M.L. (2008) *Scheduling: Theory, Algorithms, and Systems*, Springer, 3rd Edition.

[16] Sriskandarajah, C. (1993) Performance of Scheduling Algorithms for No-wait flowshops with Parallel Machines *European Journal of Operational Research*, **70**, 365–378.

[17] Tangram Technology Ltd. "Float glass production." Available from http://www.tangram.co.uk/TI-Glazing-Float%20Glass.html. Retrieved 28 November 2010.

VITA

Byungsoo Na was born in Yangsan, South Korea in October 1976. He obtained his bachelor's degree in Industrial Engineering from Pohang University of Science and Technology in 2000. Prior to continuing his education, he worked as a software engineer for three years. He joined the master program in Industrial and Systems Engineering at Texas A&M university in Fall 2004 and received a MS degree in May 2006. His master thesis topic was "the no-depot multiple traveling salesmen problem with minmax objective." He started the Ph.D. program in the School of Industrial and Systems Engineering at Georgia Tech in Fall 2006. Mr. Na's research interests are primarily in applying optimization methods to problems in supply chain and logistics. He joined a manufacturing optimization group at Samsung Electronics in January 2011.